JavaScript Handbook

3rd edition

宮坂雅輝

SOFT BANK Publishing

JavaScript Handbook

3rd edition

宮坂雅輝

SOFT BANK Publishing

はじめに

インターネットバブルもついにはじけました。しかし、それでもインターネットはわれわれにさまざまなチャンスを与えてくれます。インターネットはまだまだ発展途上なのです。これからも着実な成長を遂げていくことでしょう。

JavaScriptが登場してからすでに6年が経ちました。この間にさまざまな技術が登場しましたが、JavaScriptは廃れていくどころかますます重要な技術になっているように思います。

このような中で、Webを製作するデザイナー、プログラマーは相変わらず足りないという状況が続いています。これは、ただ単にHTMLが書けるデザイナーやプログラマー、絵が描けるデザイナーという人たちではありません。ユーザーの使い勝手を考え、インターネットの技術に精通し、「Cool!」と思わず叫びたくなるようなサイトを作れる人たちです。

いきなりこういったレベルに到達するのは難しいと思います。やはり第1歩からはじめなくてはなりません。本書がそういったレベルを目指す方のお役に立てればと思います。

この本を執筆するに当たり、真夜中にならないと仕事をしない著者の叱咤激励と、困難な編集作業、動作確認をしていただいた福井荘介様、またよりよい書籍にするべく作業をしていただいたスタッフの方々に感謝いたします。

最後に、執筆にあたりさまざまな面でサポートしてくれた妻 記子に感謝します。

2001年11月

宮坂　雅輝

CONTENTS

本書の注意事項

■表記について

本書に記載したスクリプトは、紙面の都合上、本来必要のない個所で改行されていることがあります。エラーが発生する場合は不要な改行を取り除いてみてください。本来1行なのに改行をしている部分には「⇨」を付加しています。

また、本書では書籍ページの都合によりHTMLの表記を省略している部分があります。

■サンプルスクリプトについて

本書に掲載されているスクリプトを、以下のURLで提供しております。

URL▶ http://www.neko-no-te.net/books/js3/

■対象ブラウザ

本書の対象Webブラウザと、Section3からの各サンプルに表示している動作確認欄のアイコンマークの表記は、次のようになっています。

●Windows

Webブラウザ	動作確認欄での表記
Microsoft Internet Explorer6	IE6
Microsoft Internet Explorer5/5.5	IE5
Microsoft Internet Explorer4	IE4
Microsoft MSN Explorer6	表記なし
Netscape Navigator6.2	NN6
Netscape Navigator 4.7以降	NN4

●Macintosh

Webブラウザ	動作確認欄での表記
Microsoft Internet Explorer5	IE5
Microsoft Internet Explorere4.5	IE4
Netscape Navigator6.1	NN6
Netscape Navigator4.7以降	NN4

なお、MSN Explorerは、インストールされているマシンのIEのバージョンと同じように表示されます。そのため、本書では動作確認の欄には記載しておりません。

各Webブラウザの動作確認は、次のOS環境で行っています。

OS名	Webブラウザ名
Windows XP	Microsoft Internet Explorer6
	Netscape Navigator6.2/4.78
Windows 2000	Microsoft Internet Explorer5/5.5
Windows98	Internet Explorer4
MacOS9.1	Internet Explorer5
	Netscape Navigator6.1
MacOS8.1	Internet Explorer4.5
	Netscape Navigator4.7

Section 1 OUTLINE

1-1

JavaScriptとは

JavaScriptを一言でいうならば「オブジェクト指向スクリプト言語」です。しかし、これだけでは身も蓋もありません。そこで、JavaScriptの成り立ちから見てみることにしましょう。

JavaScriptの成り立ち

JavaScriptは、Netscape社がNetscape Navigator2.0（以下NN2）のベータ版を公開したときに実装されたLiveScriptを起源にしています。これがNN2の正式版でJavaScript1.0として発表されたのが、JavaScriptのはじまりです。NN2がWebブラウザのシェアの大半を握るにしたがい、Web上でのスクリプト言語としてJavaScriptは重要な役割を担うようになりました。

この後、JavaScriptにとって大きな出来事が起こります。それは1996年の8月に、Microsoft社からInternet Explorer3.0（以下IE3）が登場したことです。このWebブラウザはWindowsの技術に特化したActiveXを導入し、さらにスクリプトでもVisual Basic Scripting Edition（以下VBScript）とともに、JavaScript1.0互換のJScript1.0を実装していました。IE3は無償ということもあり、急速にシェアを伸ばしたのです。

Netscape社は、このIE3に対抗する形でNetscape Navigator3.0（以下NN3）を1996年10月に登場させました。

NN3には、大幅な機能強化を行ったJavaScript1.1が実装されていました。この機能強化で、Webページをインタラクティブにすることができるようになりましたが、そのためにIE3に実装されているJScriptとJavaScript1.1の互換性の問題が発生しました。JavaScript1.1で強化された機能を使うとIE3では正常に動作しないのです。これ以後、Webページ制作者は互換性の問題に悩まされるようになりました。

1997年になると、インターネットはかなり一般に使われるようになりました。Microsoft社、Netscape社はそれぞれ機能を強化したInternet Explorer4.0（以下IE4）、Netscape Communicator4.0（ブラウザ部分はNetscape Navigator4.0：以下NN4）を出荷し、IE4はJScript3.0（JavaScript1.1互換）、NN4はJavaScript1.2をそれぞれ搭載しました。ただ、Java Script1.2は、NN4専用の機能拡張が多かったので、あまり一般的にはなりませんでした。

▼JScript3.0を搭載したInternet Explorer4.0



▼JavaScript1.2を搭載したNetscape Navigator4.0



その後、1998年8月にリリースされたNN4.06ではJavaScript1.3が実装されています。このバージョンではUnicodeに対応し、さらに新しいメソッドやプロパティが追加されています。

そして、1999年の3月に、Webブラウザの位置付けを変える製品が登場しました。それがInternet Explorer5(以下IE5)です。IE5は、見た目こそIE4から大きな変化はありませんが、XML、XSLに対応することにより、Webブラウザの枠に収まらず、アプリケーションのフロントエンドを担える製品になってきたのです。スクリプト面ではJScriptが強化され、アプリケーション作成に耐えられるまでに拡張されています。

さらに、2001年9月にInternet Explorer6(以下IE6)が出荷されました。ただ、Macintosh版は、まだ公開されていません(2001年10月現在)。IE6では、ECMA Script(P.6参照)やW3Cの各種仕様への対応がより進んでいます。また、Windows Media Playerとの統合も進むなど、よりブロードバンド対応となり、インターネット環境の進化に合わせたソフトウェアになっています。IE6では、JScriptに大きな進化はなく、基本的にIE5とほとんど同様です。

▼JScript5.5を搭載したInternet Explorer6



一方、Netscape社は、AOLに買収されるなど数々の激動の中、NN4の改良を細かく続けてきました。2001年10月現在の日本語版は、バージョン4.78まで公開されています。このバージョンでもJavaScript1.3が実装されています。

このNN4の改良とは別に、2000年12月にNetscape6（以下NN6）が公開されました。Netscape社はNN6を開発にするにあたり、Netscape Navigatorのソースコードを公開して、Mozillaプロジェクト（http://www.mozilla.org/）を立ち上げ、ネット上での開発を試みました。全面的なソースコードの書き換えや、W3C準拠の各種機能サポートなど野心的なプランでしたが、その完成は遅れに遅れました。

NN6は、このMozillaプロジェクトで開発されたプログラムを基に、さらに改良を行ったものです。しかし、動作スピードが非常に遅く、多くのバグが存在したことで、あまり使われませんでした。その後、動作スピードの向上と多くのバグが修正されたNetscape6.1、Windows XP対応を表明したNetscape6.2と改良されています。NN6ではJavaScript1.5が実装されています。

▼JavaScript1.5を実装したNetscape6



●WebブラウザのJavaScriptのバージョン対応

バージョン	対応Webブラウザ
JavaScript1.0	NN2、IE3（相当）
JavaScript1.1	NN3、IE4（相当）
JavaScript1.2	NN4.0〜NN4.05、IE4.5（相当）
JavaScript1.3	NN4.06〜NN4.78、IE5〜IE6（相当）
JavaScript1.5	NN6
JScript	IE3、IE4、IE5、IE6

このように、発展してきたWebブラウザ（と実装されたJavaScript）ですが、今後は、そのあり方自体が大きく変わる可能性があります。たとえば、Internet ExplorerというWebブラウザは、IE6が最後だという話があります。これは、Microsoft社がWebブラウジングの機能を完全にOSであるWindowsと統合することで、ユーザーがWebブラウザという存在を意識しなくなることを意味しているかもしれません。

この分野は、まだまだ発展途上中なので、これからも革新的な製品や状況が生まれてくることでしょう。

■ECMA Script

1996年にIE3とNN3が出荷され、それぞれに強力なJScript、JavaScript1.1が実装されたことで、JavaScriptを使ってさまざまな表現手段が可能になりました。しかし、すでに述べたように、互換性という非常に大きな問題が起こりました。どちらもJavaScriptをベースにしていたのですが、独自の拡張が行われたり、実装の不備があるなどのために、片方のWebブラウザで動いたJavaScriptが、もう一方のWebブラウザでは動かないといったことが頻繁に起こりました。そのため、Webページ制作者はページ制作時にそれぞれのWebブラウザで動作確認したり、ひどいときにはそれぞれのWebブラウザ専用に別々のページを用意する必要があったのです。

この状況に困ったWebページ制作者は、標準規格を求めるようになりました。そこでNetscape社はJavaScriptの標準化案を国際標準化団体のECMAに提出し、Microsoft社も対抗案を提示しました。ECMAは両者の標準化案をうまく融合させ、1997年7月に標準規格ECMA-262として制定しています。これはECMA Scriptと呼ばれ、ベースとなったJavaScript1.1に似たものになっています。

その後ECMA-262は、機能拡張が行われ、現在第3版（1999年12月制定）と

なっており、JavaScriptの実装にあたってはこの規格が標準となっています。IE6やNN6などは、ECMA Scriptへの準拠度が高くなっています。

ECMAについては、次のURLを参照してください。

URL▶ http://www.ecma.ch/

■JScript

ここでJScriptについて簡単にまとめておきましょう。JScriptは、Microsoft社がJavaScript互換のスクリプト言語として実装したものです。Microsoft社はスクリプト言語をIE3に実装する際に念入りにアーキテクチャを検討し、非常にスマートな形で実装してきました。それがActiveX Scriptingです。

この技術で重要なことは、Webブラウザからスクリプトの実行エンジンを切り離したという部分にあります。これにより、スクリプトエンジンだけのバージョンを上げるといったことが可能になります。たとえばIE5を使っていたとしても、スクリプトエンジンのバージョンをアップデートすると新しい機能が使えるようになります。さらに、複数のスクリプトを動作させるアプリケーション(ホストアプリケーションと呼ぶ)で、1つのスクリプトエンジンを共有することもできます。また、1つのOS上にそれぞれ異なるスクリプト言語を実行するスクリプトエンジンを用意することで、ホストアプリケーションに対して複数のスクリプト言語の使用を許可することもできます(IEでVBScriptとJScriptが動作するのはこのためです)。

2001年10月現在、JScript実行エンジンの最新のバージョンはJScript5.6となっています。Windowsのスクリプトについては、次のURLを参照してください。

URL▶ http://msdn.microsoft.com/scripting/

JavaScriptはスクリプト言語

JavaScriptは名前の示すとおりスクリプト言語です。スクリプト言語は以前からあるもので、UNIX環境では非常によく使われてきました。反復的なシステム管理タスクをひとまとめにして、システム管理を簡単にするといった目的に使われています。代表的なスクリプト言語としてはBourne Shell、Cshell、Awk、Sed、Perlなどがあります。これらのスクリプト言語とJavaScriptの大きな違いは、JavaScriptは主にWebブラウザ上で動作するものであるということです。

■JavaScriptとJavaの違い

JavaScriptには、見てわかるように「Java」という言葉が含まれています。しかし、JavaScriptとJavaはまったく関係がありません。Netscape社は、開発当初LiveScriptという名前で発表していましたが、ちょうどそのころ発表されたJavaとのマーケティング的な関係から、NN2の出荷時に「JavaScript」という名前で公開されたのです。そのため名前に「Java」という言葉が含まれていますが、開発経緯、設計思想などが異なるため、まったく別のものです。

ここで少しだけJavaについても触れておきましょう。Javaはインターネットだけにとどまらず、昨今のコンピュータ業界の中で非常に重要な役割を占めるようになっています。このJavaにはプログラミング言語としての側面と、実行環境としての側面があります。通常のプログラミング言語、たとえばC言語、C++言語、Pascalなどといったものは実行環境ごとに(要するにそれぞれのOSごとに)コンパイラが用意され、コンパイルした後の実行ファイルは各OSごとにしか動作しません。それに対してJavaは、Java VM(Virtual Machine)上で動作するという実装にしたため、どのようなOS上であってもJava VMさえ存在すれば実行できます。つまり、どのOS上であっても同じJavaのプログラムファイルで実行できることになります。

Javaを大きく分けると、Webブラウザ上で動くJavaアプレット(Applet)、クライアントJavaプログラム、サーバーサイドJavaプログラムの3つになります。Webページ制作者にとっては、JavaアプレットとサーバーサイドJavaプログラムに注目する必要があるでしょう。

Javaアプレットは、Webサーバーからアプレットと呼ばれる小さなJavaプログラムをダウンロードしてWebブラウザ上、またはOS上のJava VMの上で動作します。しかし現状では、Java VMの起動・動作が遅いためJavaアプレットを使うことはほとんどありません。

サーバーサイドJavaプログラムにはいくつかの種類があるのですが、この中でもWebサーバーの機能拡張として使われているServletやEJB(Enterprise Java Beans)が注目されています。これらはJava言語の生産性の高さや、安定性から、最近の電子商取引の基盤技術として使われはじめています。

少なくともJavaとJavaScriptがまったく異なるものであるということがわかっていただけたでしょうか?

■JavaScriptの特徴

JavaScriptを使うことにより、どのようなことができるのでしょうか? さまざまなインターネット関連技術の中でも、JavaScriptは重要な役割を担っています。まずJavaScriptはWebブラウザの中で動作するため、WWW(World

Wide Web)を使っているほとんどの人の環境で動作します。さらに、制作にあたって特別なツールを必要とすることもなく、単純なテキストエディタがあれば作成できます。またスクリプト言語であるため、コンパイルといった作業も必要ありません。JavaScriptは、数あるスクリプト言語の中でも、非常に敷居の低いものであるといえます。

JavaScriptはWebブラウザ側、つまりクライアント側で動作します。これは、Webサーバーの管理者でなくても、Webページにさまざまな機能の追加を、簡単に行えることを意味し、CGIなどと比較して、非常に大きなメリットとなっています。そのため、ほとんど同じことをするのであれば、CGIよりもJavaScriptで実装させたほうがよいことになります。ただし、当然JavaScriptにはできないことがあるので、実際にはCGIと使い分けるのがもっともよい制作スタイルであることは間違いありません。

■JavaScriptではできないこと

ここで、JavaScriptではできないことを挙げておきましょう。大きくいうと2つあります。

1つ目は、セキュリティの関係から、Webブラウザの動作しているマシンから情報を取り出したり、書き込んだりすることができません(Cookieは除く)。これは、WebブラウザでWebページにアクセスしたときに、勝手にハードディスク内からデータが取り出されるといったことや、逆にウイルスなどをハードディスク内に書き込まれないための仕様です。

2つ目は、JavaScriptはWebページを見る側からコードを隠すことができないことです。JavaScriptはHTML文書内に記述されているため、Webブラウザでページを表示した人は簡単にコードを見ることができます。Webページ上で、ある機能をどのようにして実現しているかを知られたくないときには、JavaScriptではなく、他の方法を使うのがよいでしょう。

こういった制約があるため、JavaScriptはクライアント側でのフォームのための事前処理や、ダウンロードしたWebページの要素をインタラクティブに動作させるといった用途に向いています。

1-2

JavaScriptの展開

前のセクションで紹介したように、JavaScriptには非常に簡単ながら、かなり強力なプログラムの作成能力があります。この点に注目して、さまざまな場面でJavaScriptを使おうという試みが行われています。

DHTMLでのJavaScript

IE、NNの両者には、Dynamic HTML（以下DHTML）という機能が実装されています。これは名前は同じものなのですが、IEとNNとの間に互換性はほとんどありません。しかし、両者とも目指す方向性は同じです。それは、GIFアニメーションやJavaアプレットを使わずに、Webページ内の各HTML要素を、再読み込みなしに動かそうというものです。

このDHTMLがどのように実装されているかというと、Cascading Style Sheets（以下CSS）と呼ばれるスタイルシートの各要素の属性値を、スクリプト言語によりダイナミックに変化させるという方法が取られています。そのため、スクリプト言語が重要になっており、とりわけIEとNNの両方で動作するJavaScriptの重要性が増しています。

WebサーバーサイドでのJavaScript

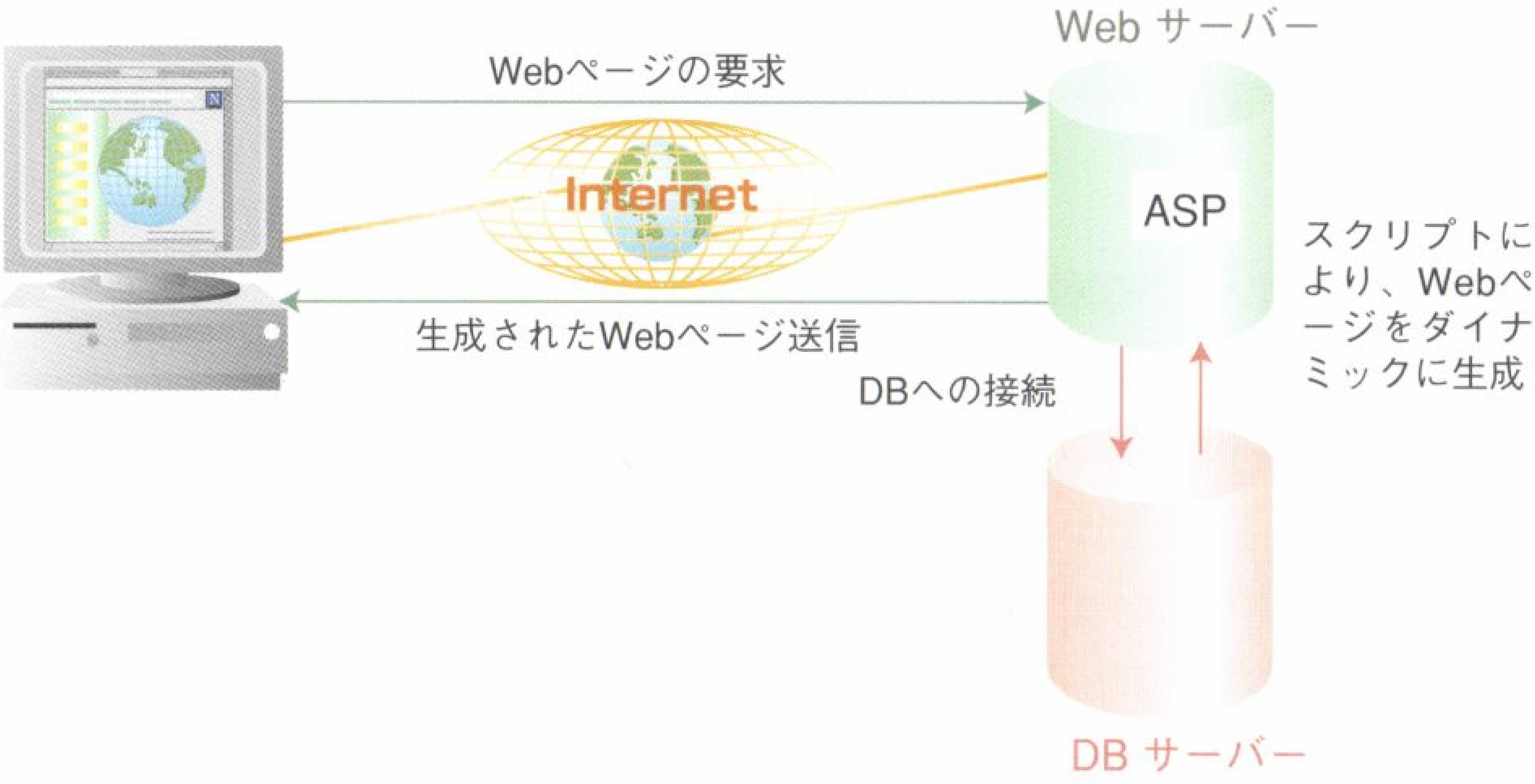
JavaScriptには、Webブラウザというクライアント環境で動作するイメージがありますが、実際にはWebサーバー側でもかなり使われてきています。Webサーバー側のスクリプト言語というと、すぐに思い付くのがCGIに使われているPerlです。Perlは非常に強力なのですが、CGI上で使うという点と、さまざまな記述で同じ機能が実現できる柔軟性があるため、書く人によってコードがバラバラになってしまう傾向があります。

こういった状況の中で、Webサーバー側でスクリプトを動作させてしまおうという製品がいくつか登場しています。たとえば、Microsoft社のActive Server Pages（以下ASP）といったものです。ASPは、WebサーバーであるInternet Information Server（以下IIS）とともに無償であるため、最近ではさまざまなWebページで使われるようになってきました。

ASPとは、HTML文中にサーバーサイドで動作するスクリプトを記述し、そのページの要求がきた時点でスクリプトを実行し、ダイナミックにページを生成し

て送り出すものです。このスクリプト記述用の言語として、VBScript、JScriptが標準で実装されています。

▼ASPの動作概念図



ASPではデータベースへの接続も非常に簡単に行うことができるので、イントラネット構築などでは非常に重要な役割を占めるようになっています。また、Microsoft社の.NET戦略の中の1つのテクノロジーであるASP.NETも、JScriptにより記述することができます。

デスクトップへの展開

JavaScriptは、もはやWWWの世界では、なくてはならないものです。逆に一般には、WWW上で使うものという認識もあります。しかし、最近では、JavaScriptは思わぬところで使われていたりします。たとえば、WindowsではWindows Scripting Host(以下WSH)により、JavaScriptはWindowsのスクリプト言語として使えるようになっています。

WSHとはWindows上でのスクリプト言語実行環境です。Webページ内でJavaScriptを用いるのと同様のことを、Windows上でも行うことができます。さらにWSHは、さまざまな組み込みオブジェクトを実装し、Windows全体のコントロールも可能になっています。そのため、簡単なことならば、Visual Basic(以下VB)などでプログラムを作成しなくても、WSHで実現できるようになります。

WSHではIE上でのスクリプト言語の実装状況と同じように、Active Scriptingに対応するスクリプト実行エンジンが用意されているスクリプト言語

ならば動作します。標準ではVBScript、JScriptが組み込まれています。

なお、WSHはWindows 98/Me/2000/XPに搭載されています。Windows 98以前のWindows 95/NT 4.0の場合には、Microsoft社から無償でインストールファイルが提供されています。次のURLからダウンロードしてください。

URL▶ http://www.microsoft.com/japan/developer/scripting/

そしてすべての環境へ

スクリプト言語はWebブラウザからクライアント環境へと、使用範囲を広げてきました。今後、さらにその範囲を広げようとしています。Windowsの世界ではWSHによりJScriptなどのスクリプト言語でWindowsのすべてをコントロールできるようになっています。たとえば、スクリプト言語でADSI(Active Directory Service Interface)やWMI(Windows Management Interface)を操作すると、ユーザー管理、マシン管理、システム管理などといったシステム管理のほとんどのことが実現できるようになります。

そして、Microsoft SQL Server7.0/2000でも、データ操作言語などにスクリプト言語が使えるようになっているため、データベースの分野においてもJavaScript(正しくはJScript)が使われています。

さらにJavaScriptは、Webサービスを実現する際にも重要な役割を担うと思われます。Webサービスは、クライアントからの要求に対して、単にWebのページを返すだけでなく、その問い合わせに対して何らかの処理をした結果を返したり、サービスを行ったりすることです。Webサービスは、XML(eXtensible Markup Language)をメッセージングのフォーマットとして使い、システム間の連携を容易にします。Webサービスを実現する技術にはさまざまなものがありますが、先ほどのASPやASP.NETのようにJavaScriptで記述できるものが多数あります。

このようにスクリプト言語、その中でもJavaScriptは重要な存在で、いずれすべてのコンピューティング環境で使われることになるでしょう。

2-1

JavaScriptの組み込み

このセクションでは、JavaScriptの記述にあたり必要な情報を、わかりやすく解説しています。まずは、JavaScriptの組み込みからです。

JavaScriptの組み込み方

Webページに JavaScriptを組み込むのは非常に簡単です。JavaScriptに限らず、Webページにスクリプトを組み込むときには、以下のように<script>タグを用いて記述します。

```
<script language="使用する言語">
    スクリプト文
</script>
```

組み込むスクリプト言語がJavaScriptなら、次のように記述します。

```
<script language="javascript">
    スクリプト文
</script>
```

なお、language属性はHTML4.0では非推奨属性となっており、将来はなくなる予定です。HTML4.0以降ではlanguage属性の代わりにtype属性を用いて、以下のように記述します。

```
<script type="text/javascript">
    スクリプト文
</script>
```

ただし、type属性は現在のところすべてのWebブラウザに対応しているわけではありません。そのため、本書では従来どおりのlanguage属性を用いて解説をしていきます。

JavaScriptの書き方

JavaScriptを書くときには、次の点に注意してください。

- 「;」が命令の区切りとなる
- 大文字と小文字を区別する

■JavaScriptでの1命令文

VBなどでは、1行につき1つの命令といった規則がありますが、JavaScriptにはそのような制限がありません。しかし、何の規則もないとどこからどこまでが1つの命令なのか区別がつかなくなってしまいます。そこで、「;」(セミコロン)を命令と命令の区切り文字として使います。

JavaScriptの命令は、複数の命令を1行に記述してもかまいません。逆に、1つの命令を複数の行に分けて記述してもかまいません。ただし、このときに気を付けなくてはならないのは、命令や変数名の単語の途中で改行をしてはいけないということです。

```
document.write("Hello");
alert("Hello");
```

上記の記述を、1行に書き換えると次のようになります。

```
document.write("Hello"); alert("Hello");
```

逆に、1行で記述できる命令を複数の行にも分割できます。

```
win1 = window.open('',  'newwin', 'toolbar=no');
```

上記の命令は、以下のように記述することもできます。

```
win1 = window.open('', 'newwin',
'toolbar=no');
```

KNOWLEDGE

■大文字と小文字

JavaScriptで用いる変数名やキーワード、メソッド名やプロパティ名は、それぞれ大文字・小文字を区別します。そのため、大文字・小文字を間違っているとエラーとなります。

Basic系の言語やスクリプトでは、大文字と小文字の区別をしなくてもよいので、これらの言語に慣れている方は、注意する必要があります。

■JavaScriptを記述する位置

JavaScriptを実装するための<script>タグは、HTMLファイルのどの位置に記述すればよいのでしょうか？ 多くのWebページでは<head>タグの中に配置しているようです。しかし、フォームボタンに関連付いたスクリプトなどは、それぞれの要素の近くに記述するとメンテナンスがしやすいといった場合があります。<script>タグはHTMLファイル中に何回でも記述することができるので、好みに応じて使うことができます。ここで、気を付けなくてはならないのは、スクリプトの呼び出しよりも先にスクリプトの本体を記述しておく必要があるということです。

たとえば、Webページ読み込み時に動作させたいスクリプトは、<body>タグのonload属性による呼び出しよりも先に記述しなくてはなりません。

コメント

JavaScriptのコードの中には、コメントを入れることができます。コメントを利用してスクリプト内にコードの解説を入れることにより、スクリプトのメンテナンスが容易になります。コメントを入れる方法は2種類あります。

- // コメント
- /* コメント */

まず、「//」によりコメントを記述する方法です。この「//」は、記述した部分から行末までをコメントとして扱います。つまり、1行のコメントを入れるときに使います。C++言語やJava言語などで使われるコメントの実装と同じです。

```
<html>
<title>コメントの実装1</title>
```

```
<body>
<script language="javascript">
<!--
   // この行はコメントとして扱われます。
   document.write("Hello JavaScript World!");
//-->
</script>
</body>
</html>
```

次は、「/∗」と「∗/」でコメントを囲む方法です。囲んだ範囲がコメントとして扱われますので、間に改行を入れることも可能です。これは主にC言語などで使われるコメントの実装と同じ方法です。

```
<html>
<title>コメントの実装2</title>
<body>
<script language="javascript">
<!--
   /∗  複数行に
      わたるコメントにも
      対応します。 ∗/
   document.write("Hello JavaScript World!");
//-->
</script>
</body>
</html>
```

外部ファイルを用いて実装

HTMLファイルに直接JavaScriptを記述する方法は、すでに述べたとおり、記述性、メンテナンス性がよくなるのですが、不都合が生じることもあります。

たとえば、次のような場合には、できればHTMLファイルに直接JavaScriptを記述したくはありません。

●非常に長いスクリプト
●さまざまなHTMLファイルで同じスクリプトを使う
●JavaScriptのコードを見せたくない

そのため、JavaScriptではスクリプトだけを書いた外部ファイルを用意し、それを呼び出して実装するということもできます。外部ファイルの読み出しには<script>タグのsrc属性を用います(値は相対パス、もしくは絶対パスで指定)。

```
<script language="javascript" src="****.js">
</script>
```

なお、JavaScriptのコードを書いた外部ファイルの拡張子は「.js」としなくてはなりません。

■組み込みのサンプル

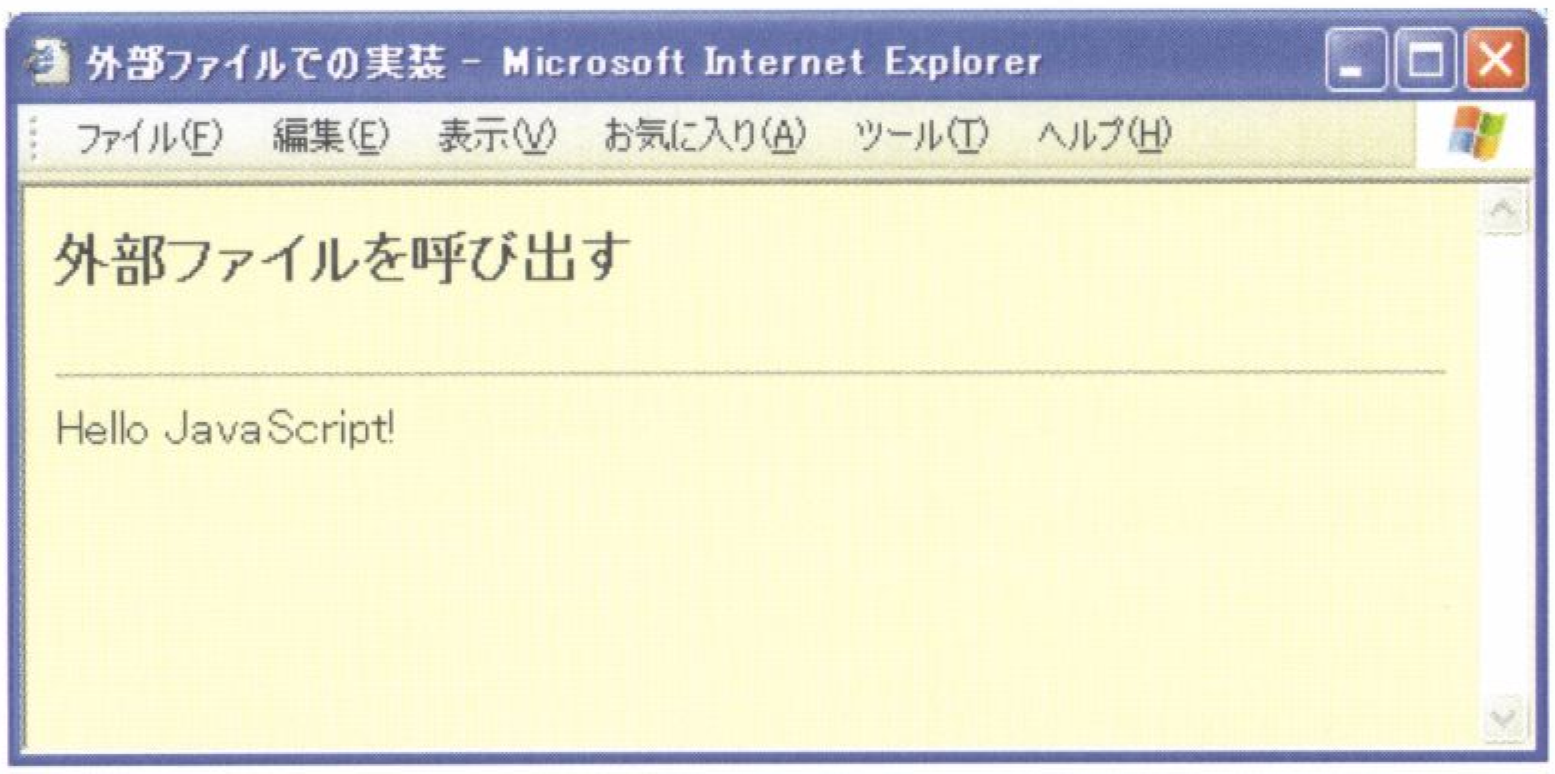
外部ファイルによる実装サンプルを挙げます。外部スクリプトファイルを呼び出すHTMLファイルは、次のようになります。

```
<html>
<title>外部ファイルでの実装</title>
<body bgcolor="#fff8dc">
<h3>外部ファイルを呼び出す</h3>
<hr />
<script language="javascript" src="ex_write.js"></script>
</body>
</html>
```

そして呼び出されるJavaScriptのほうは、「ex_write.js」という名前で次のようなコードを記述したファイルを作ります。

```
document.write("Hello JavaScript!");
```

このサンプルを表示させると次のようになります。



■外部ファイル組み込みの注意点

外部ファイルによるJavaScriptの実装は、JavaScriptをサポートしていないWebブラウザからスクリプトを自動的に隠せるといったメリットがあります。しかし、HTMLファイルの他に別のファイルをダウンロードする必要があるため、Webサーバーへの負荷が高くなり、処理の遅いWebサーバーや、高負荷状態のWebサーバーに置くと問題が起こる可能性がある点に注意してください。

また、外部ファイルによるJavaScriptの実装では、HTMLファイルにはスクリプトを記述しないのでHTMLファイルソースとしては隠せます。しかし、外部ファイルはいったんWebページをブラウジングしているユーザーのPC上にダウンロードされるため、完全にユーザーから隠せるわけではありません。

なお、この外部ファイルによる実装はNN2、IE3では動作しません。

■外部ファイル組み込みに必要なWebサーバーの設定

外部JavaScriptファイルを使うと、Webサーバーから拡張子が「.js」のファイルをダウンロードします。しかし、WWWサーバーの初期設定ではJavaScriptファイルをダウンロードできないようになっている場合があります。そこで、サーバー側でJavaScriptファイルのダウンロード要求に対処できるように設定する必要があります。

現在、Webサイトを提供しているインターネットサービスプロバイダ(以下ISP)の大半は、UNIX上にApacheというWebサーバーを実装し、サービスを行っています。ISPですでに外部ファイルの取り込みができるように設定されている場合には問題ありませんが、それ以外の場合は自分で設定するしかありません。Webサーバーの管理者ならば、httpd(Webサーバーのプログラム名)の設定を、ユーザーなら「.htaccess」というファイルを編集します。

では、Apacheの場合の設定方法を記述します。

●httpdの設定を変更する方法

JavaScriptファイルが扱えるようにMIMEタイプを追加します。方法は2種類あります。

・「<インストールしたディレクトリ>/conf」の下にある「mime.types」に、以下の1行を追加します。

```
application/x-javascript js
```

・「<インストールしたディレクトリ>/conf」の下にある「srm.conf」に以下の1行を追加します。

```
AddType application/x-javascript .js
```

ファイル内で追加する場所は、設定ファイルを見て、同様の記述の前後に記述するとよいでしょう。

●「.htaccess」を編集する方法

JavaScriptファイルの置かれているWebサーバー上のディレクトリに「.htaccess」というファイルを用意します。UNIXでは「.」(ピリオド)がファイル名の最初に付くと、隠しファイルとして扱われます。このファイルに、以下のような記述を追加します。

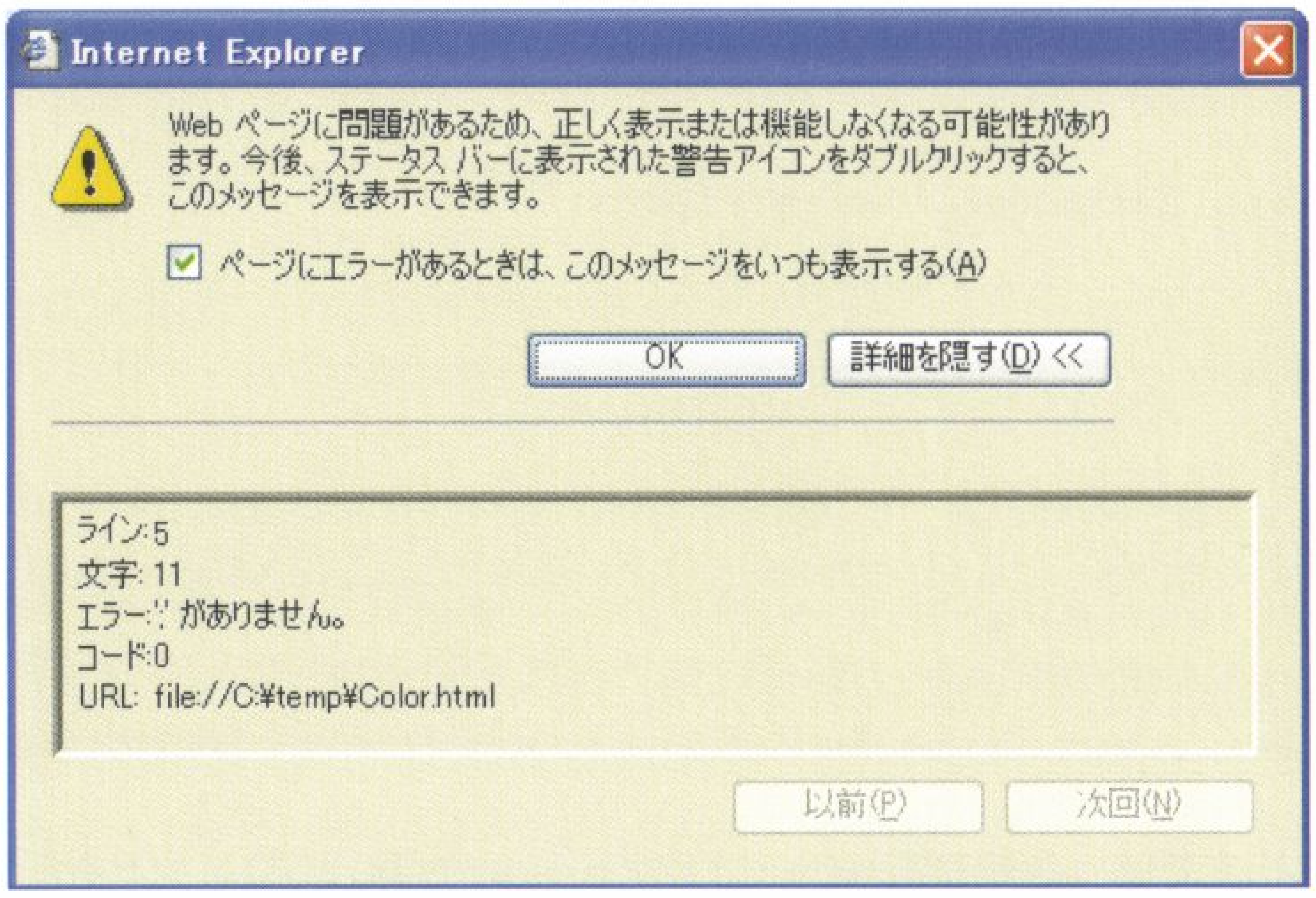
```
AddType application/x-javascript .js
```

この設定を行っても、JavaScriptファイルがダウンロードされない場合は、ISPのWebサーバーがApacheでないか、ISPが「.htaccess」を使うのに必要な設定を行っていない可能性がありますので、ISPに問い合わせてください。

エラーについて

JavaScriptを組み込んだWebページ内に、JavaScriptの文法のエラーがあった場合はどうなるのでしょうか？　次のような警告ウィンドウが表示されます。

▼IEでのエラーメッセージ



エラーの表示方法はWebブラウザによって異なります。Webブラウザによってはエラーが存在すると思われる行が表示される場合もありますし、ステータスバーにエラーがあるとだけ表示されるものもあります。エラーと思われる行が表示されている場合は、その行までにエラーがあるということを示しているので、その行までを注意深く確認することで、素早くエラーを発見することができます。

JavaScriptに対応していないWebブラウザのための記述

JavaScriptは、現在の主要なWebブラウザで動作します。しかし、Webブラウザの中にはJavaScriptが動作しないものがありますし、家電やPDAなどに組み込まれているWebブラウザでは、JavaScriptが動作しないものが多数存在します。

Webページを作成する場合、これらのJavaScriptが動作しないWebブラウザにも配慮する必要があります。JavaScriptが動作しないWebブラウザのためにできる対応は2つあります。

- ●スクリプトのコメント処理
- ●<noscript>タグの記述

■スクリプトのコメント処理

JavaScriptが動作しないWebブラウザには、「スクリプト行をコメントのように見せる」という方法を用います。

```
<html>
<title>文字を表示する　コメント対応</title>
<body>
    <script language="javascript">
    <!--
    document.write("Hello JavaScript World!");
    //-->
</script>
</body>
</html>
```

通常HTML文では、注釈宣言(<!-- コメント -->)を用いてコメントを記述しますが、これを応用してJavaScriptに対応していないWebブラウザではスクリプト全体をコメントとして認識するようにします。HTML文のコメントの指定と異なるのは、コメントの終了部の前に「//」が入っていることです。この「//」はJavaScriptではコメント行として認識されます。これがないとJavaScriptの実行エンジンは「-->」を解釈できない命令として認識するためエラーとなってしまいます。そのため、JavaScriptをWebページに実装するときには次のような形で組み込むのが一般的です。

```
<script language="javascript">
<!--
    スクリプト
//-->
</script>
```

JavaScriptの動作するWebブラウザは、<script>タグを認識してスクリプトを実行しますが、<script>タグを認識できないWebブラウザはコメントタグに囲まれた部分を解釈せず、スクリプトを実行しません。

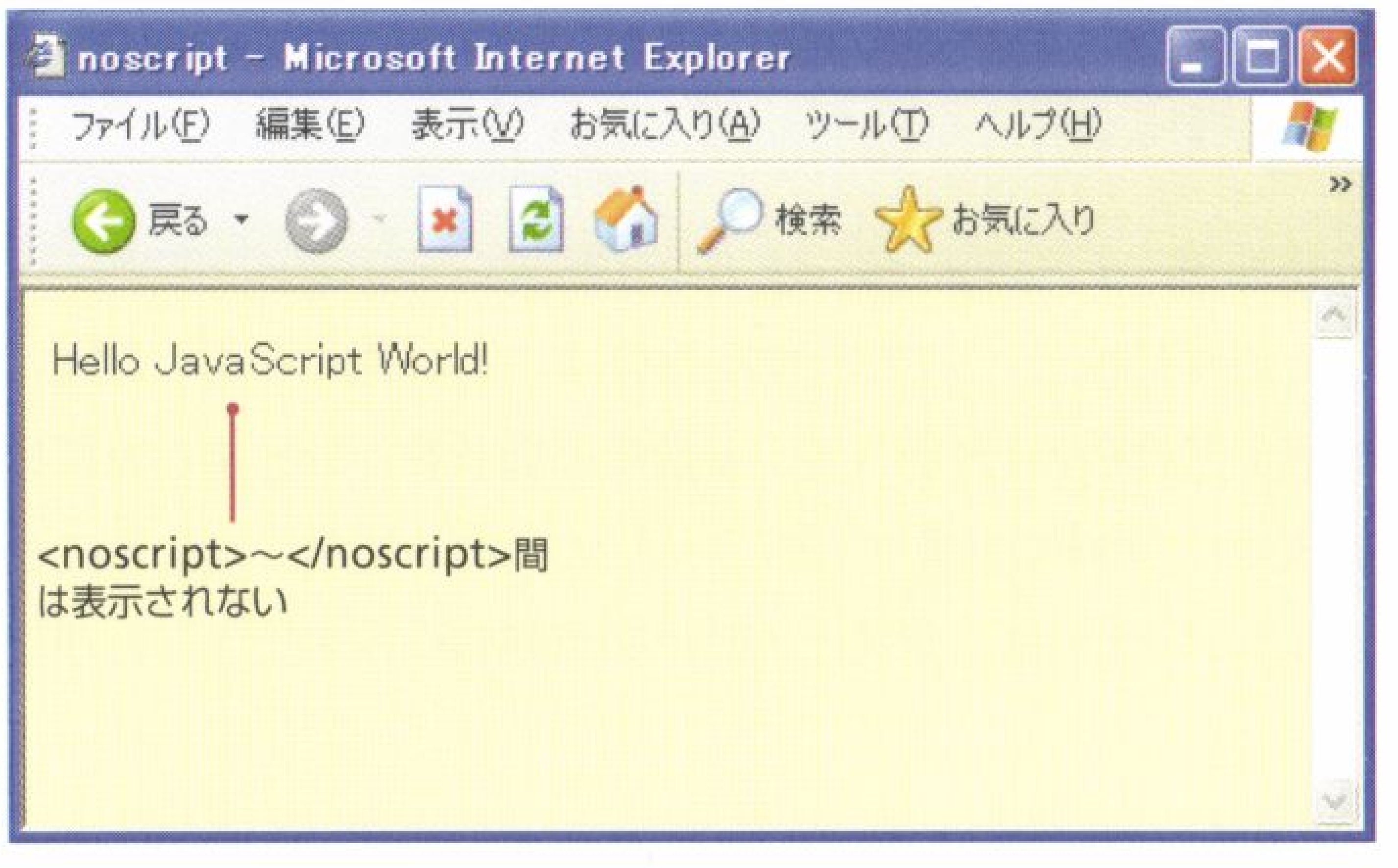
■<noscript>タグ

スクリプトをコメント行で囲むことにより、JavaScriptに対応していないWebブラウザがスクリプトの処理をしないようにできます。ただ、それだけではJavaScriptが動作しないWebブラウザでは、何も処理を行わないようにしただけです。JavaScriptが動作しないWebブラウザに対しても、代わりのメッセージを表示するなど、何らかの配慮をしておく必要があります。

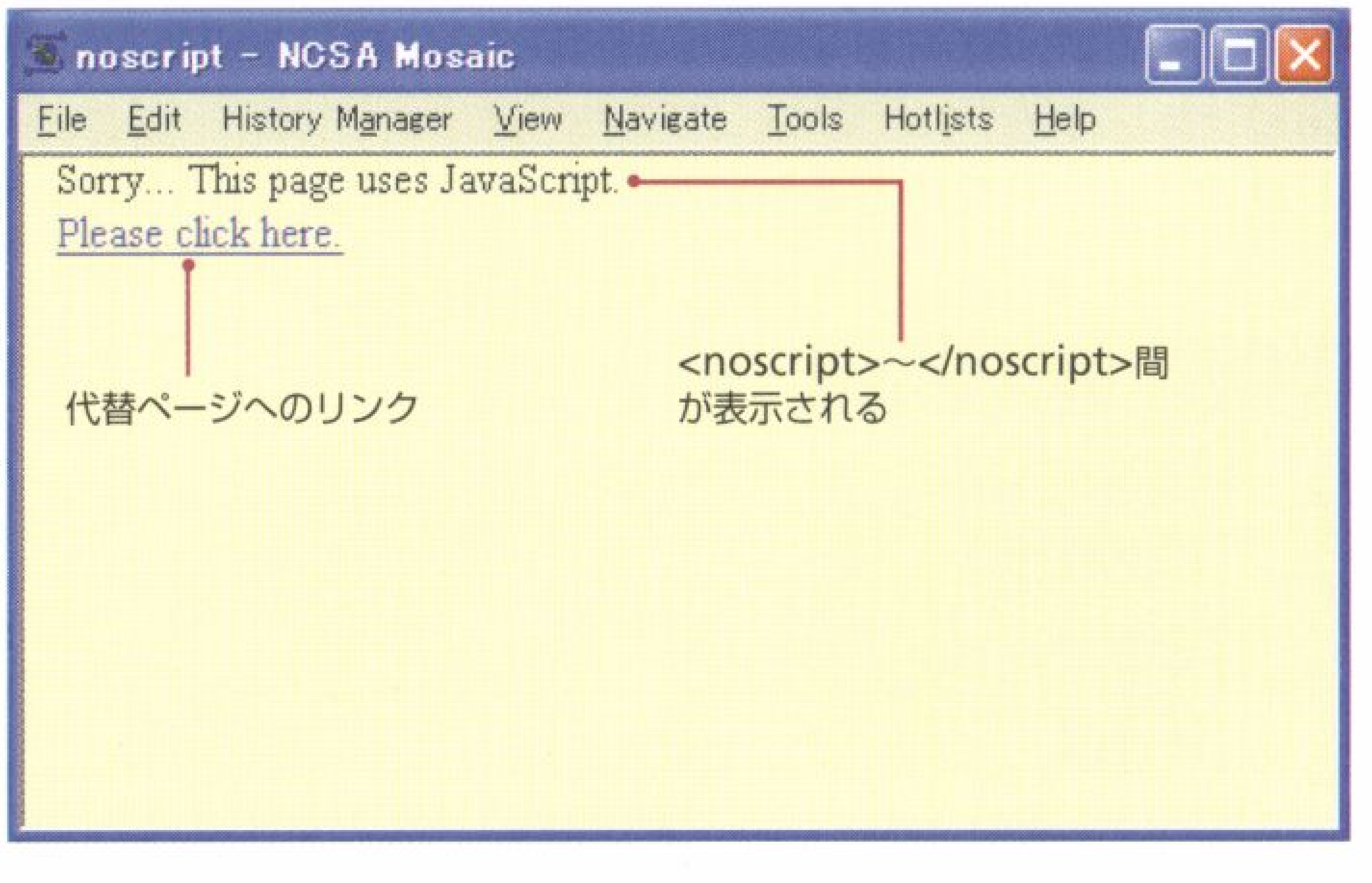
このときに用いるのが<noscript>タグです。<noscript> ～ </noscript>の間に記述した内容は、JavaScriptに対応したWebブラウザでは無視されます。次の記述をJavaScript対応ブラウザと非対応ブラウザで表示してみましょう。

```
<script language="javascript">
<!--
   document.write("Hello JavaScript World!");
//-->
</script>
<noscript>
   Sorry... This page uses JavaScript.<br />
   <a href="../noscript.html">Please click here.</a>
</noscript>
```

▼JavaScript対応ブラウザでの表示



▼JavaScript非対応ブラウザでの表示



2-2 JavaScriptで扱えるデータ

スクリプトではさまざまな情報を扱います。たとえば、文字列、数字といったものです。JavaScriptでは、情報を次の 4 つの形式に分類して扱います。

●JavaScriptで扱えるデータ型

データ型	例
数値	10、0.1、10e3など
文字列	"Hello"、"Fun"など
論理値	true（真）またはfalse（偽）のどちらか
null	null値。何も入っていないことを表す特別なキーワード

他の言語に比べると扱えるデータの型が少ないのですが、あまり高度で複雑な処理をしない限り問題となることはないでしょう。

数値

JavaScriptの数値型は、他の言語と比較すると非常に少ないです。これは、Javaなどの言語では細かく分かれている数値のほうを、ひとまとめにしているからです。このため、複雑な計算を行おうとした場合は問題が起こることがあります。JavaScriptの数値型は、整数と浮動小数点数の両方を扱うことができます。

■整数

「整数」とは小数点以下を持たない数値のことです。整数には正と負があり、表現できる最大の整数は、動作させるOSやWebブラウザによって異なります。またJavaScriptでは、10進数、8進数、16進数の3種類の整数を扱うことができます。例としては次のようなものがあります。

例(整数)	種類
123	10進数(正)
-123	10進数(負)
056	8進数
0x7F	16進数

■浮動小数点数

「浮動小数点数」とは、小数点以下を持ち、小数の表現が浮動小数点により表されているものです。浮動小数点数にも正と負があり、さらに小数点以下の精度はOSやWebブラウザにより異なります。浮動小数点数では小数点以下を持った10進数、またはべき乗の識別子(E、e)と指数を持った10進数を取り扱えます。

例(浮動小数点数)	種類
1.23	10進数(正)
-1.23	10進数(負)
1e3	べき乗の識別子と指数を持った10進数

文字列

文字列とは0文字以上の文字のことをいいます。文字列は「"」(ダブルクォート)または「'」(シングルクォート)で囲んで記述します。

例(文字列)	種類
"Hello"	ダブルクォートで囲む
'JavaScript'	シングルクォートで囲む
"234"	数字を囲む
""	空文字

ダブルクォートやシングルクォートで囲むと、数字でも文字列として扱われます。最後の囲まれた文字のない「""」は、「空文字」と呼ばれるものです。

論理値とnull値

論理値は、「true」(真)または「false」(偽)のどちらかを持っている値のことをいいます。これはデータの比較や判定などに用います。JavaScriptではC言語と異なり、「1」、「0」といったものは論理値として扱いません。

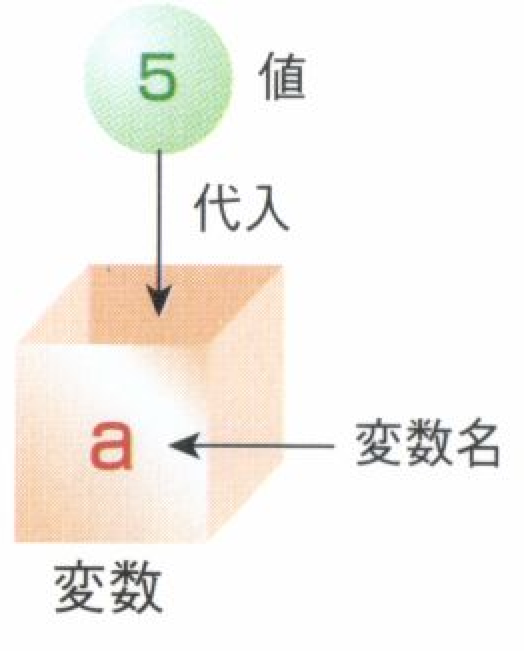
null値は、「何もない」ことを表す特別な値です。たとえば、定義されていない変数を参照しようとした場合などにnull値が返されます。null値は「何もない」という値を持っているため、文字列のところで出てきた空文字や数値の0とは異なります。JavaScriptでは、null値は「null」と表現します。

2-3 変数

変数の生成

JavaScriptで、「2-2　JavaScriptで扱えるデータ」で解説したデータを扱うには、変数を用意し、それにデータを入れるという手段を取ります。変数には名前があり、ある値を変数に入れると変数名で値を参照することができます。すなわち、ある値を入れることのできる名前の付いた箱が変数といえます。変数に、ある値を入れることを代入といいます。

▼変数に、ある値を入れるのが代入



JavaScriptでは、変数にどんな型のデータでも格納することができます。この変数を作ることを「変数の宣言」と呼びます。

変数の宣言は次のように記述します。

```
var 変数名;
```

変数の宣言は、「var」の後に変数名を記述して行います。varは「variable」(変数)という意味です。実際に用いると次のようになります。

```
var sample;
```

上記のように記述すると、まだ値を持っていない「sample」という名前の変数が作成されます。

変数の宣言と同時に値を代入するには、次のように記述します。

```
var sample="Hello";
```

上記のように記述すると、「sample」という名前の変数を作成し、その中に「Hello」という値(文字列)を代入します。JavaScriptでは、ある変数をはじめて使ったときに、その変数の宣言が行われたと解釈するようになっているので、varによる変数の宣言を行わず、次のように記述したとしても、同様の結果を得ることができます。

```
sample="Hello";
```

変数に値を代入するには等号(=)を使います。これは代入演算子と呼ばれます(P.29参照)。

変数の名前

プログラム中では、さまざまな処理を行うために複数の変数を用います。そのため、変数を何らかの方法で区別する必要があります。変数に名前を付けるときには、いくつか気を付けなくてはならないことがあります。

- ●大文字と小文字を区別する
- ●アルファベット、数字、アンダースコア（_）で構成される
- ●1文字目はアルファベットか、アンダースコア（_）
- ●予約語を使わない

■大文字・小文字の区別と文字構成

JavaScriptでは大文字と小文字を区別します。変数もプロパティ名やメソッド名と同様に、大文字と小文字を区別しています。この区別はUNIXの世界では当たり前のことなのですが、大文字と小文字の区別のないWindowsに慣れていると、ちょっと面倒に感じるかもしれません。

大文字と小文字が区別されるので、次の変数名はそれぞれ別のものとして認識されます。

- ●SAMPLE
- ●Sample
- ●sample

変数は、アルファベット、数字、アンダースコア(_)を用いて構成します。ただし、変数の1文字目だけは、アルファベットかアンダースコア(_)しか用いることができない点に注意してください。

■予約語

JavaScriptでは、いくつかのキーワードを、JavaScriptが使うものとして予約しています。これを予約語といいます。予約語は変数名、関数名として使うことができません。

●予約語一覧

break	case	catch	continue	default	delete	do
else	finally	for	function	if	in	instance
of	new	return	switch	this	throw	try
typeof	var	void	while	with		

また、ECMA-262では将来予約語になる可能性のあるキーワードとして次のものを挙げています。これらも、変数名や関数名に使わない方がよいでしょう。

●将来の予約語一覧

abstract	boolean	byte	char	class
const	debugger	double	enum	export
extends	final	float	goto	implements
int	interface	long	native	package
private	protected	public	short	static
super	synchronized	throws	transient	volatile

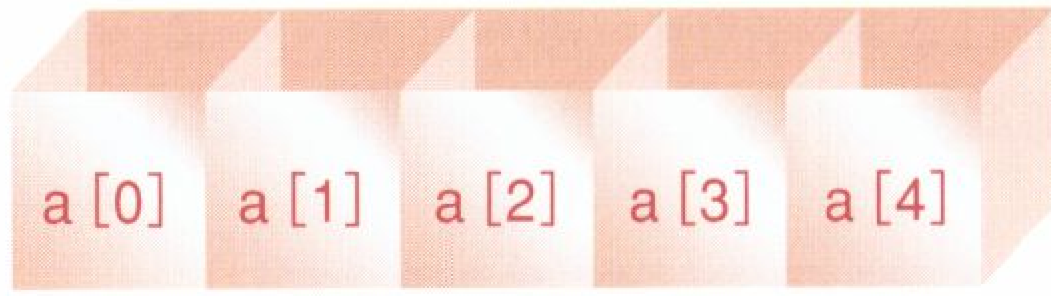
配列

配列とは、複数個の変数をかたまりとして扱える型にしたものです。表現としては、変数に添え字が付いたものです。配列の宣言は次のようになります。

```
a = new Array(5);
```

このように宣言すると、5個の値を代入することができる「a」という配列ができます。この配列の要素は、a[0]、a[1]、a[2]、a[3]、a[4]の5つになります。P.26の変数の図で考えると、次のように5つの箱があることになります。

配列名 a



気を付けなくてはならないのは、配列の一番最初の添え字は「0」からはじまるということです。これは、JavaやC言語などの経験のある方ならわかると思いますが、そうではない方にはピンとこないかもしれません。非常に間違いやすいので注意してください。

配列は、数値を使って変数を操作する場合に便利です。また場合によっては配列を使うことでスクリプトを簡単にできるため、マスターするとスクリプトの制作効率を上げることができます。

変数の処理

変数はただ値を入れるだけでは意味がありませんが、変数に対して何らかの処理を加えれば、さまざまなことを実現できます。ここでいう処理とは、変数、演算子を用いて何らかの計算を行うことです。演算子には、代入演算子、算術演算子、論理演算子、比較演算子、条件演算子、文字連結演算子といったものがあります。JavaScriptの演算子の種類や使用方法は、JavaやC言語の演算子とほとんど同じです。

■代入演算子

代入演算子は、ある値を変数に代入するために使います。基本的には、代入演算子の右側に記述されている値や式の結果などを、左側に記述されている変数に代入する処理を行います。以下に主な代入演算子を示します。

例	内容
a = b	aにbを代入する
a += b	aにa+bを代入する
a -= b	aにa-bを代入する
a *= b	aにa*bを代入する
a /= b	aにa/bを代入する
a %= b	aにa/bの剰余を代入する

■算術演算子

数値計算を行う演算子です。和(+)、差(-)、積(＊)、商(/)を求める演算子の他に、割り算の余りを求める剰余演算子(%)があります。

例	内容
a + b	aとbを足す
a - b	aからbを引く
a ＊ b	aとbを掛ける
a / b	aをbで割る
a % b	aをbで割った余り

なお、商演算子で割る数(b)を「0」にすると、演算結果は無限大(Infinity)が返されます。さらに、剰余演算子で割る数(b)を「0」にすると、その剰余は計算できないため「NaN」(Not a Number)が返されます。これらは、計算上でトラブルを引き起こす可能性があるので、気を付けてください。

また、上記の演算子以外にも、インクリメント演算子(++)、デクリメント演算子(--)、単項否定という演算子があります。

例	内容
a = b++	aにbを代入してから、bの値を1増やす
a = ++b	bの値を1増やしてから、aに代入する
a = b--	aにbを代入してから、bの値を1減らす
a = --b	bの値を1減らしてから、aに代入する
a = -b	aにbの符号を反転させたものを代入する

■比較演算子

比較演算子は、右側と左側の値を比較するときに用います。比較演算子を使った式は条件式として用います。主な比較演算子を以下に示します。

例	内容
a == b	aとbが等しいときに真
a != b	aとbが等しくないときに真
a > b	aがbよりも大きいときに真
a >= b	aがb以上のときに真
a < b	aがbよりも小さいときに真
a <= b	aがb以下のときに真

■論理演算子

論理演算子は右側、左側の条件式を評価して、それぞれ成り立っているかどうかでtrue(真)、false(偽)を返します。これは複数の条件式を評価するために用いることが多いです。論理演算子を以下に示します。

例	内容
条件式a && 条件式b	aとbが成り立っていたら真(AND)
条件式a ll 条件式b	aとbのどちらかが成り立っていたら真(OR)
!条件式a	aが真の場合には偽、aが偽の場合には真(NOT)

■条件演算子

条件演算子は、これまで見てきた演算子とは少し異なります。これは条件式と2つの値から構成され、2つの値のうちのどちらかが返される値になります。書式は以下のようになります。

```
(条件式) ? 値1 : 値2
```

条件式の部分は、評価すると論理値(trueまたはfalse)が返るような式にします。条件式が真ならば、この式全体が返す値が「値1」に、偽ならば「値2」になります。例としては次のようなコードになります。

```
<script language="javascript">
<!--
   var sample = 10, tmp;
   tmp = ( sample>=5 )? "sampleは5以上" :
⇒     "sampleは5より小さい";
   document.write(tmp);
//-->
</script>
```

上記のコードの場合、変数sampleの値を評価し、その結果が真であるため「sampleは5以上」と表示されます。

■文字列連結演算子

文字列連結演算子(+)は、文字列同士を連結する演算子です。書式としては次のようになります。

```
文字列1 + 文字列2
```

上記のように記述すると「文字列1」と「文字列2」が連結されます。実際に用いると以下のようになります。

```
chars = "Hello " + "JavaScript";
```

上記のコードでは、変数charsに「Hello JavaScript」という連結された文字列が代入されます。

■演算子の優先順位

演算子は、同時に用いたときにどちらが先に実行されるかという優先順位があります。複雑な式を記述する場合には意識する必要があります。

演算子の優先順位は以下の順序で評価されます。



1.括弧（()）
2.積（＊）、商（/）、剰余（%）
3.和（＋）、差（－）
4.比較演算子（<、<=、>、>=）
5.等価演算子（==、!=）
6.論理積（&&）
7.論理和（||）
8.条件演算子（?）
9.代入演算子（=、+=、-=、*=、/=、%=）

■演算の順序の制御

すでに解説したように、演算子には優先順位があります。優先順位は、通常の数学の計算と同様に、乗算・除算が加算・減算よりも優先されます。加算、減算を乗算、除算よりも先に計算したい場合には「( )」(括弧)で囲みます。また、剰余演算は乗算、除算と同じ優先順位です。次のような計算があったとします。

```
result = 3*(4+5);
```

この場合、括弧で囲んでいる「4+5」を計算して、その結果に「3」を掛けることになります。

正規表現

正規表現とは

JavaScriptで文字列を扱うことは、つい最近までかなり面倒でした。というのは、Perl、Awkなどといった UNIXの世界における標準的なスクリプト言語で実装されている正規表現(Regular Expression)が、JavaScriptには実装されていなかったからです。そのため文字列処理を行うときには、1文字ずつ識別したり、substrメソッドで文字を切り出してから処理を行ったりと、結構面倒な処理を記述しなくてはなりませんでした。しかし、JavaScript1.2(JScript3.0)から正規表現に対応したことで、文字列の検索といったことはかなり容易に実現できるようになりました。

正規表現とは、文字列がどのようなパターンになっているかを形式的に表すものです。たとえば、「Th」からはじまるもの(This、Theなど)は、「Th*」と表現します。このように正規表現は、通常の文字(a～zなど)と、「メタキャラクタ」という特殊文字から構成される文字列のパターンです。

正規表現は、文字列のパターンを表すものなのですが、これだけでは正規表現を生かすことはできません。そのためJavaScriptには、正規表現を活用するためのオブジェクトと、いくつかのメソッド、プロパティが用意されています。

ここでは正規表現と、それらを活用するオブジェクト、メソッド、プロパティについて解説します。

正規表現の作成

正規表現では、通常区切り文字でパターンを囲みます。JavaScriptでは、「/」(スラッシュ)を用いて次のような書式で表現します。

```
/パターン/
```

またJavaScriptでは、もう1つパターンを指定する書式があります。それはRegExpオブジェクトを使って指定する方法です。

```
var obj = new RegExp("パターン", "フラグ");
```

フラグには、後述する"g"、"i"を指定できます(P.36参照)。

どちらの方法を使ってもまったく同じです。Perlなどで正規表現に慣れている人は、「/」でパターンを囲む指定方法を、VBScriptなどで正規表現を使っていた人はRegExpオブジェクトを用いればよいでしょう。本書では原則として「/」を用いた書式を使います。

パターンは多種多様な指定ができます。以下にそれぞれ指定の仕方を解説していきます。

■1文字にマッチする

マッチする文字が、ある特定の1文字の場合には、正規表現として、その文字自体を使います。たとえば「a」が含まれる文字列をマッチさせたいときには、次のように記述します。

```
/a/
```

なお、「/ /」はマッチ演算子と呼ばれるもので、ある文字列が正規表現で表現されたパターンにマッチするかどうかを調べる演算子です。

また、ある1文字と任意の1文字をマッチさせたいときには、「.」(ピリオド)を使います。たとえば「a」ではじまる文字列を探したいときには、次のように記述します。

```
/a./
```

これにより「ab」、「at」、「az」といったものがマッチしたと認識されます。

これ以外に、ある複数の文字の中のどれかにマッチすればよいという場合があります。このようなときに使うのが「文字クラス」です。文字クラスの書式は、次のようになります。

```
[文字列]
```

たとえば、「a」、「b」、「c」のうちのどれか1文字が入っていればよい場合には、次のように記述します。

```
[abc]
```

また、「-」を使って、その間の記述を省略することもできます。たとえば、何らかの数字1文字にマッチする場合には、次のように記述します。

```
[0-9]
```

さらに、「含まれていない場合」という否定的な指定をする場合には、「^」を使います。たとえば数字以外の1文字にマッチしたい場合には、次のように記述します。

```
[^0-9]
```

これ以外に、「略記法」による指定もできます。主な略記は以下のようになります。

●主な略記法

略記法	文字クラス	意味
¥w	[a-zA-Z0-9_]	アルファベット、数字、下線
¥W	[^a-zA-Z0-9_]	アルファベット、数字、下線以外
¥s	[¥r¥t¥n¥f]	空白文字（スペース、復帰文字、タブ、改行文字、ラインフィード）
¥S	[^ ¥r¥t¥n¥f]	空白文字以外
¥d	[0-9]	数字
¥D	[^0-9]	数字以外

■複数の文字にマッチする

ここまでは、1文字にマッチする場合を紹介してきました。しかし、これだけではなく、複数の文字列にマッチするかどうかを確認したい場合があります。たとえば、「img」ではじまり、後に数字が付いている文字列をまとめてマッチさせたい場合には、次のように記述します。

```
/img[0-9]*/
```

ここで気が付くのはアスタリスク「*」です。これは「0回以上にマッチする」という意味です。たとえば上記の例であれば「img」、「img8」、「img123」などにマッチします。このような記号を「量指定子」と呼びます。量指定子には次のようなものがあります。

●量指定子

記号	マッチ回数
*	0回以上にマッチ
+	1回以上にマッチ
?	0回、1回にマッチ
{m}	ちょうどm回にマッチ
{m,}	m回以上にマッチ
{m,n}	m回以上n回以下にマッチ

■マッチする位置を指定する

マッチを調べたい場所が、行の先頭だけとか行の末尾だけといった場合があります。このような場合には、「位置指定子」を用いてマッチする位置を指定します。これは、単語を調べるなどといったときに重宝します。

●位置指定子

位置指定子	意味
^	文字列(行)の最初にマッチする
$	文字列(行)の最後にマッチする
¥b	単語の境界にマッチする
¥B	単語の境界以外にマッチする

■マッチする条件を指定する

マッチを調べるときに、1つだけマッチするものを探すのか、マッチするものをすべて探すのかといったことを指定したい場合があります。また、通常は大文字と小文字は区別されますが、大文字と小文字を区別したくない場合もあります。このときには、以下のような修飾子で条件を指定できます。

●条件を指定する修飾子

修飾子	意味
g	マッチするものすべてを見つける
i	大文字と小文字を区別しない

指定方法としては正規表現の後ろに付けます。

```
/a./g
```

このように指定すると、検索対象文字列から「ab」、「az」、「ac」などをすべて取り出します。

正規表現を使う

正規表現のパターンを指定する方法は、すでに解説しました。では、指定したパターンを用いて実際に検索などを行うには、どのようにしたらよいのでしょうか？　正規表現を用いる方法には大きく分けて2つあります。1つは文字列オブジェクトに実装されているmatchメソッド、replaceメソッド、searchメソッドなどを用いる方法です。もう1つはRegExpオブジェクトが持っているメソッドなどを用いる方法です。

文字列オブジェクトのメソッドを用いる方法は、簡単に正規表現を扱うことができます。文字列オブジェクトの中に次のようなメソッドがあります。

●文字列を検索し、マッチした文字を返す

```
srtObj.match(regExp)
```

●マッチした文字列を置換する

```
srtObj.replace(regExp, repStr)
```

●文字列を検査し、マッチした位置を返す

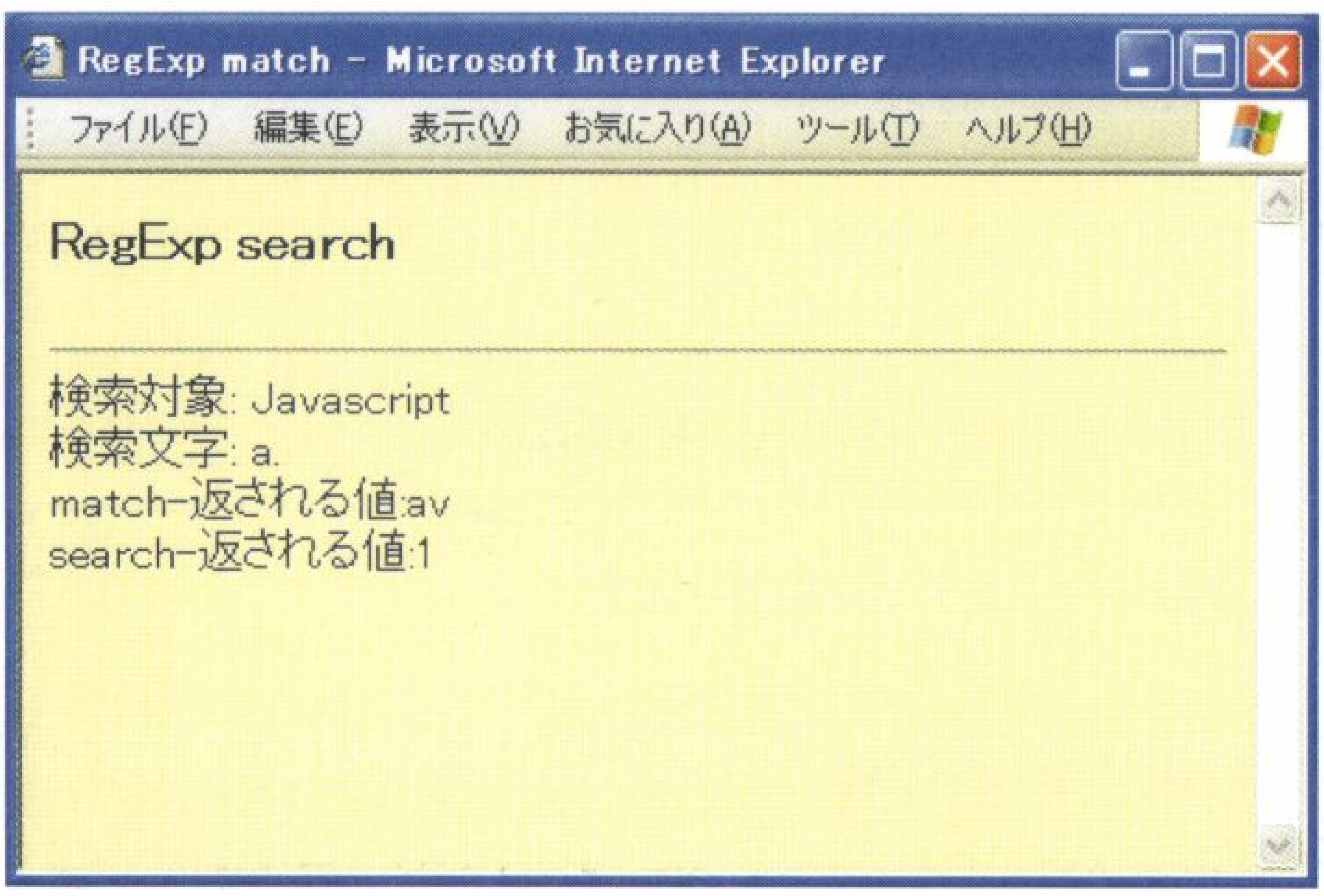
```
srtObj.search(regExp)
```

これらはどれも、パターンを引数に取り、マッチ処理、または置換処理を行います。マッチ処理のサンプルは次のようになります。

```
<html>
<title>RegExp match</title>
<body bgcolor="#fff8dc">
<h3>RegExp search</h3>
<hr />
<script language="javascript">
```

```
    var myReg = /a./;
    var str = "Javascript";
    document.write("検索対象: " + str + "<br />");
    document.write("検索文字: " + myReg.source +
        "<br />");
    strMatch = str.match(myReg);
    strSearch = str.search(myReg);
    document.write("match-返される値:" + strMatch +
        "<br />");
    document.write("search-返される値:" + strSearch);
</script>
</body>
</html>
```

このサンプルを実行すると、次のように表示されます。



条件によって処理を変える

ここまでで式や論理式の記述方法を解説してきたので、これらを使ってよりプログラムらしいスクリプトを記述してみましょう。

ある式(比較式など)が条件式の条件部分になると説明しました。また条件演算子は条件の真、偽という2つの値のどちらが返されるかという簡単な条件選択を実現することができます。しかし、これらの方法では上から下へと流れる直線的なプログラムの流れを変えることはできません。そこで、条件によって処理を変えたり、繰り返しの処理を行ったりする方法を解説します。

条件分岐

if~else文(条件文)を式と組み合わせて使うと、条件に応じて処理を分岐させ、プログラムの流れを変更することができます。if~else文には何パターンかの記述方法があります。まず、一番簡単なifを用いた文は次の方法で記述します。

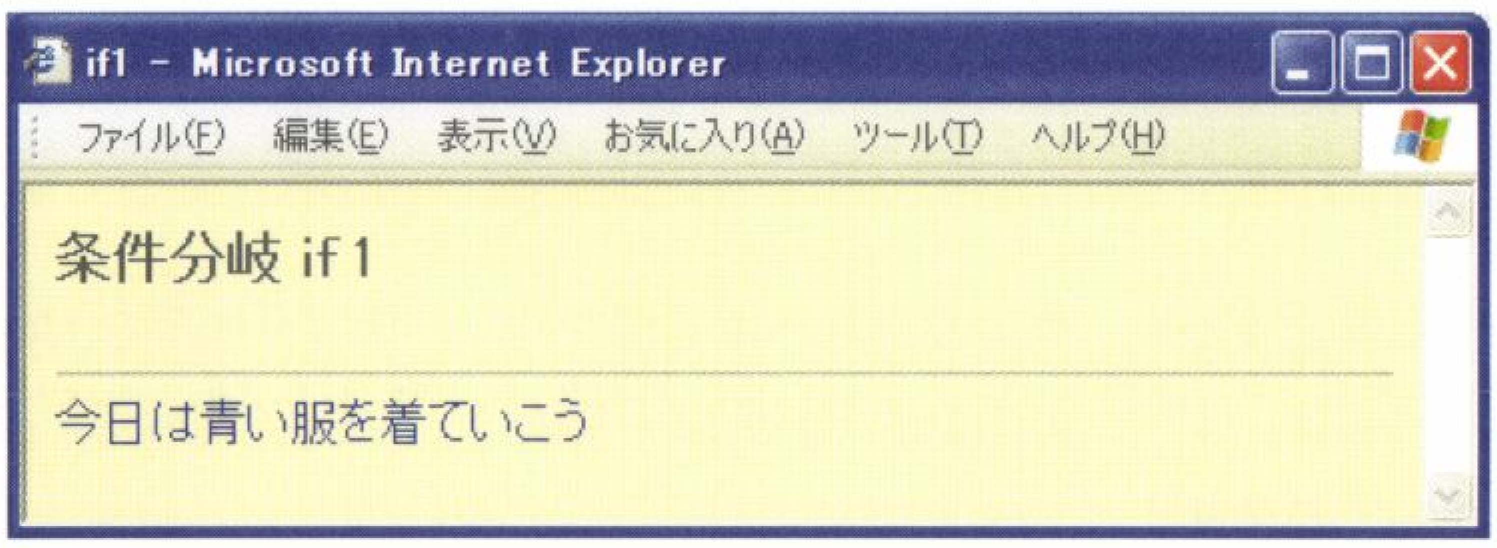
```
if( 条件式 )
    処理;
```

このように記述すると、条件が真ならば処理を行い、偽なら処理を実行せずに次の行へ移ります。また、この処理の部分は中括弧({ })を用いて処理を複数実行させることもできます。

```
if( 条件式 ){
    処理1;
    処理2;
    ……;
}
```

逆に処理が1つならば中括弧を省略できます。if文を用いたサンプルは次ページのようになります。このサンプルでは条件が真となるので、次ページの図のように中括弧内の処理が行われています。

```
<html>
<title>if1</title>
<body bgcolor="#fff8dc">
<h3>条件分岐 if 1</h3>
<hr />
<script language="javascript">
<!--
   var myColor="blue";
   if(myColor=="blue"){
      document.write("<font color='blue'>今日は青い服
         を着ていこう</font>");
   }
//-->
</script>
</body>
</html>
```



もしも上記サンプルで変数「myColor」の値を「blue」ではなく「red」とすると、条件式が偽となってしまうため何の処理も行われません。

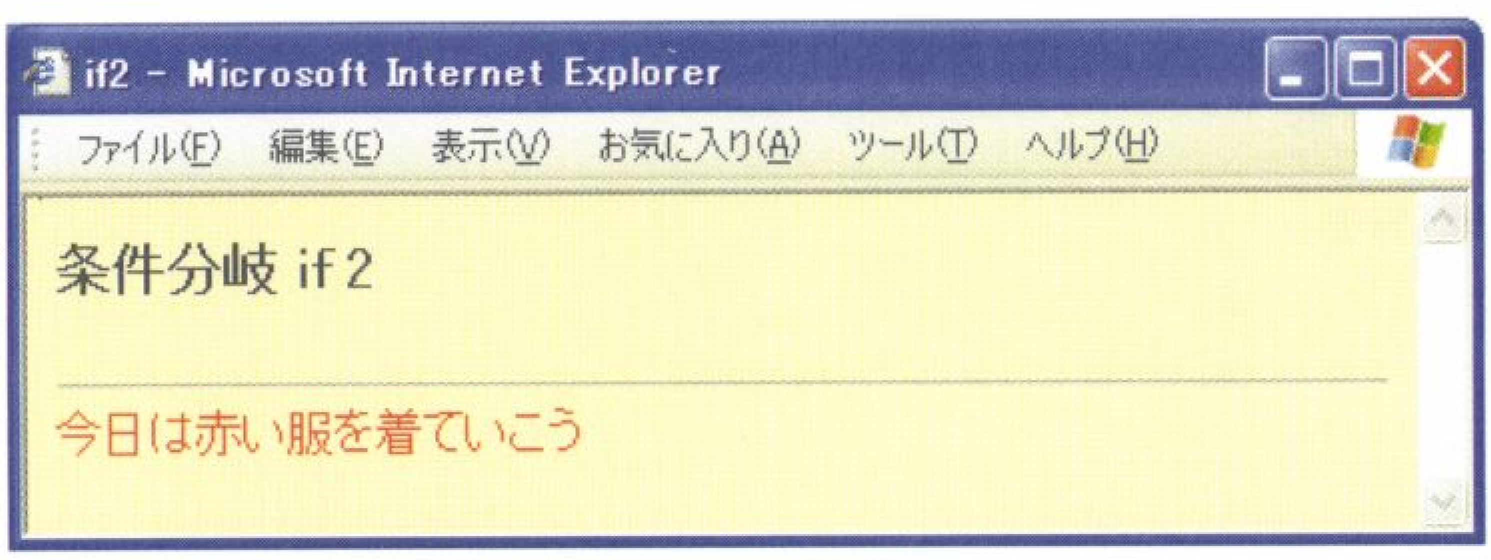
このように単純な分岐はifだけでも記述できますが、条件式が偽のときにも何らかの処理を行いたい場合は、次のようにelseも用います。

```
if( 条件式 ){
   処理1;   ←―――――― 単数でも複数でもかまわない
}
else {
   処理2;   ←―――――― 単数でも複数でもかまわない
}
```

この場合、条件が真のときには処理1が、偽のときには処理2が行われます。if～else文を用いたサンプルは次のようになります。

```
<html>
<title>if2</title>
<body>
<h3>条件分岐 if 2</h3>
<hr />
<script language="javascript">
<!--
   var myColor="red";
   if(myColor=="blue"){
      document.write("<font color='blue'>今日は青い服
⇒        を着ていこう</font>");
   }
   else{
      document.write("<font color='red'>今日は赤い服
⇒        を着ていこう</font>");
      }
//-->
</script>
</body>
</html>
```

このサンプルでは条件式が偽となります。そのためelseに記述されている処理が実行されます。実行結果は次のようになります。

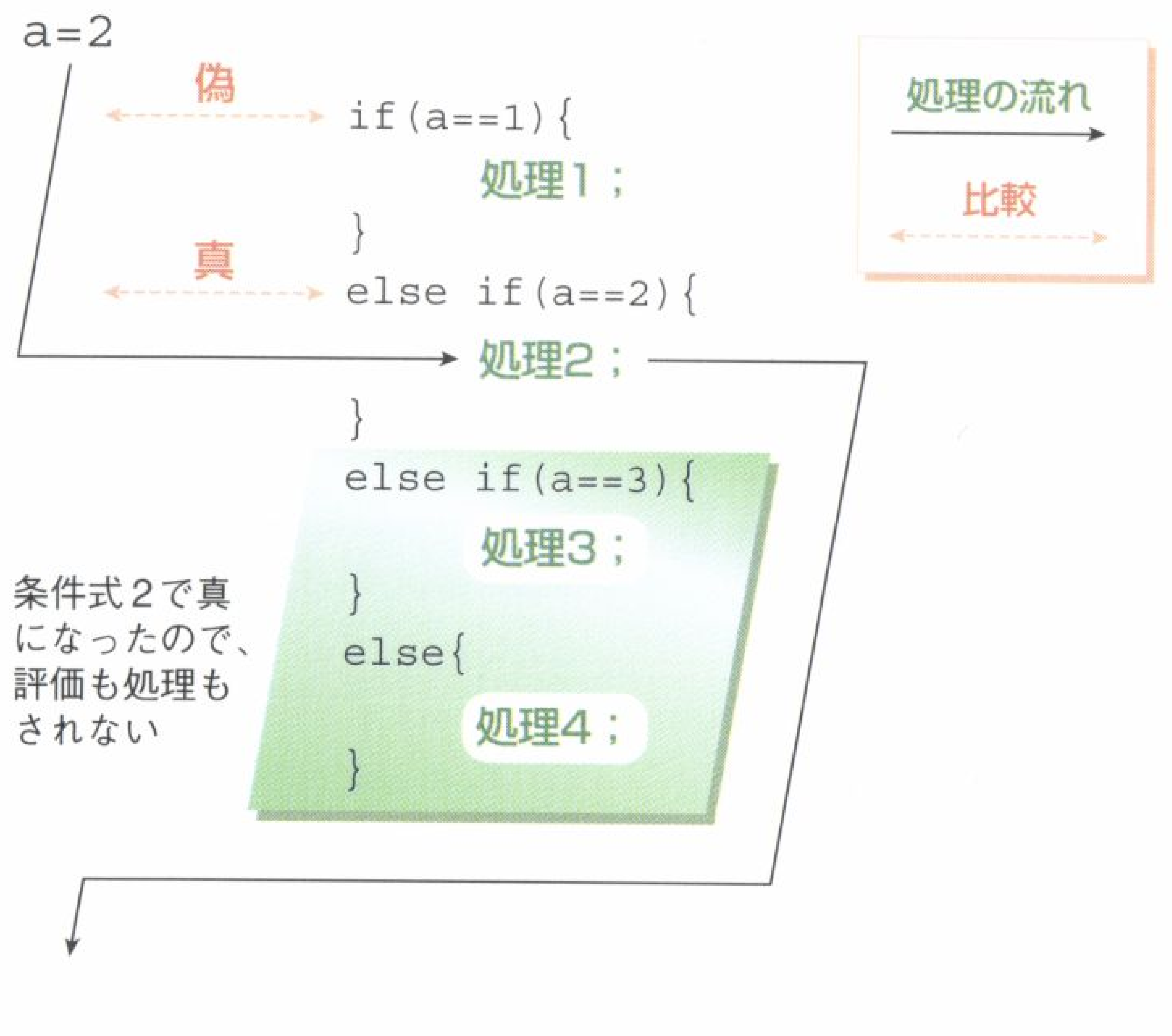


現実にはもっとたくさんの分岐をしたい場合もあります。その場合にはifとelseを組み合わせて、さらに複数の条件式を評価できるようにします。

```
if( 条件式1 ){
    処理1;
}
else if( 条件式2 ){
    処理2;
}
else if( 条件式3 ){
    処理3;
}
        …
else{
    処理n;
}
```

このように記述すると任意の数の条件を評価することができるようになり、複雑な条件分岐が行えるようになります。しかし、上記の記述を使う場合に気を付けなくてはならないことがあります。たとえば、条件式1で真となると、処理1が実行された後、if～else文で書かれたコード全体を抜けてしまうことです。

▼真になった時点で条件式を抜ける



この図でもわかるように、if～else文では、ある条件式で真になったらその部分の処理を行い、それ以降の条件式は評価も実行もしません。

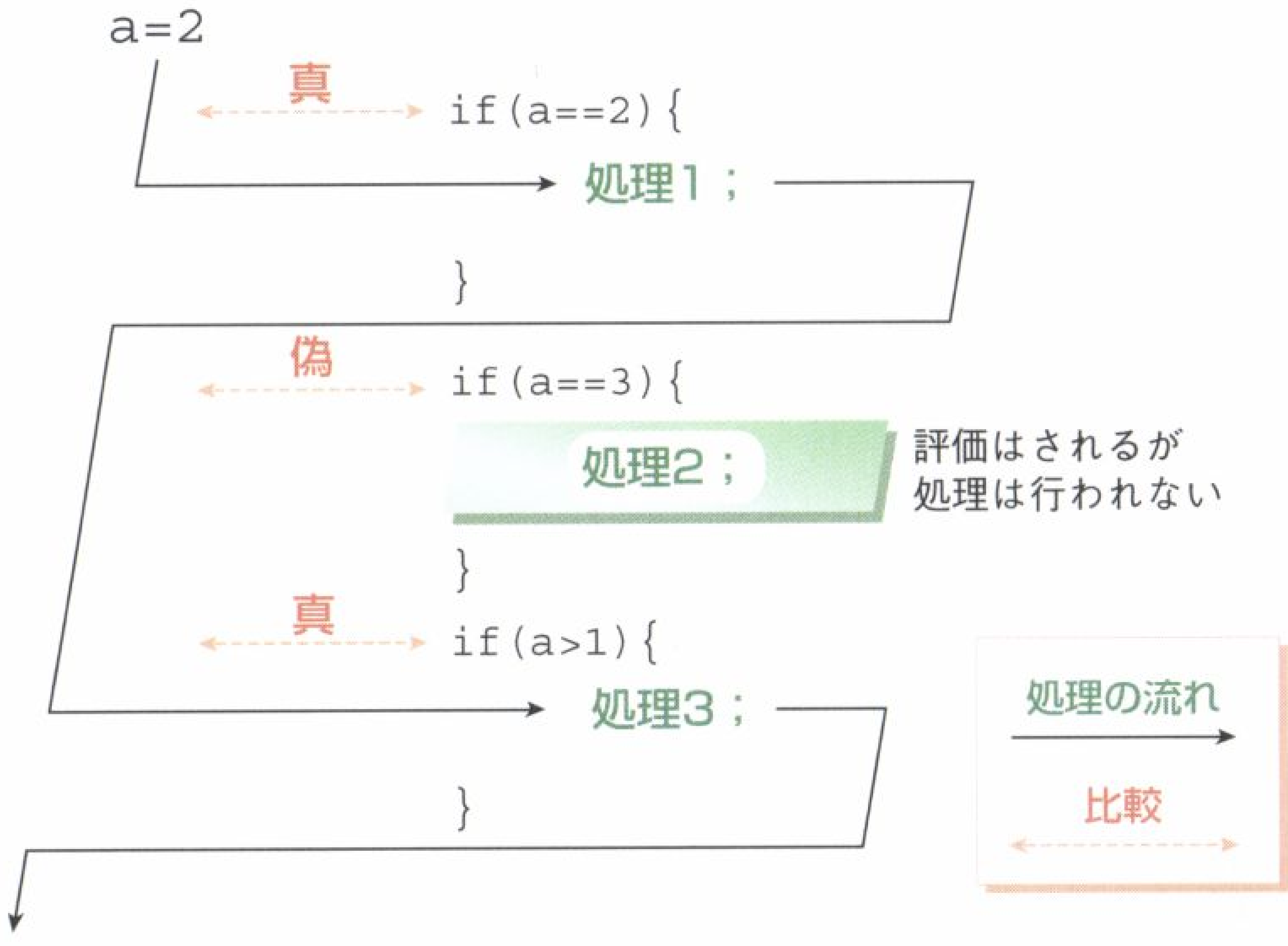
それでは、書いた条件式をすべて評価してほしい場合には、どうしたらよいのでしょうか？　解決法は簡単です。次のように記述します。

```
if( 条件式1 ){
    処理1;
}
if( 条件式2 ){
    処理2;
}

……
```

このように記述すると条件式をすべて評価し、真になった部分の処理をすべて行うことになります。

▼記述する条件式をすべて評価する



繰り返し処理

これまでに簡単なプログラミングを実現するif〜else文を解説しました。しかし、これらはスクリプトの行がそれぞれ1回しか実行されませんでした(if〜else文の場合には実行されない行もあります)。しかし、プログラミングをしていると、同じ行(処理)を複数回繰り返したいという状況が発生します。これを実現するのが繰り返し処理です。この処理を「ループ」と呼びます。ループには2種類あります。1つは条件に基づいてその条件が満たされるまで繰り返すもの、もう1つは設定された範囲を超えるまで繰り返すものです。

■for文を使ったループ

for文を使ったループはプログラミングの世界では非常に基本的なもので、C言語をはじめ、Basic、Perl、Javaなどにも同様の仕組みが実装されています。

for文を使ったループは、基本的に指定した回数だけ処理を繰り返すのに用いられることが多いです。

for文は次のように記述します。

```
for( 初期状態 ; 終了条件 ; 継続処理 ){
    処理;
}
```

サンプルとしては、次のように記述することができます。

```
for( i=1 ; i<=10 ; i++ ){
    処理1;
}
```

これは、初期状態は「iが1」、終了条件は「iが10になったとき」、繰り返しごとに「iに1を足していく」というfor文です。これによりiが1から10になるまで「処理1」を10回繰り返すことができます。このときiを「カウンタ」と呼び、数を数えるのに用います。また、条件はループの最初に評価されます。継続処理は、ループ内の処理が終わり次のループに入る前に実行されます。次は、for文を使ったサンプルになります。

```
<html>
<title>for 1</title>
<body bgcolor="#fff8dc">
<h3>for文で繰り返し処理が行われます</h3>
<hr />
<script language="javascript">
<!--
for(i=1;i<=7;i++){
   document.write("<font  size="+i+">Hello  JavaScript!
⇨     </font>< br />");
}
//-->
</script>
</body>
</html>
```

このスクリプトを実行すると、次のように表示されます。



もしもfor文を使わずにこのWebページを書いたとしたら、非常に面倒なことになるのは容易に想像できると思います。他にも文例を挙げておきましょう。

●iを0からはじめ、10より小さい間、iを1ずつ加えながら繰り返す

```
for( i=0 ; i<10 ; i++ )
```

●iを10からはじめ、20以下の間、iを1ずつ加えながら繰り返す

```
for( i=10 ; i<=20 ; i++ )
```

●iを10からはじめ、0より大きい間、iを1ずつ減らしながら繰り返す

```
for( i=10 ; i>0 ; i-- )
```

●iを0.0からはじめ、1.0以下の間、iを0.2ずつ加えながら繰り返す

```
for( i=0.0 ; i<=1.0 ; i=i+0.2 )
```

■for文を使ったループでの注意

for文は非常に便利なものですが、プログラミング次第では深刻なエラーを引き起こします。for文の後ろの「( )」の中に条件を書くのですが、ここで終了条件を誤るといつまでたってもループが終わらず、スクリプトが終わらないということが起きます。これを「無限ループ」と呼びます。

無限ループはプログラム上有効な場合もあるのですが、意図しない場合は深刻なエラー以外の何ものでもありません。このようなエラーを避けるため気を付けなくてはならないのは、終了条件の記述の仕方です。終了条件で使える演算子は、次の4種類です。

```
<    >    <=    >=
```

これ以外の演算子ではほとんどエラーとなりますので、使わないようにしましょう。また条件の設定は必ず真になるものにしてください。

■for～in文を使ったループ

for文を使ったループは一般的なものですが、JavaScriptではさらに拡張を行い、for～in文という仕組みを実装しています。for～in文を使ったループはオブジェクトのすべてのプロパティに自動的にアクセスするために使われます。for～in文の記述は以下のようになります。

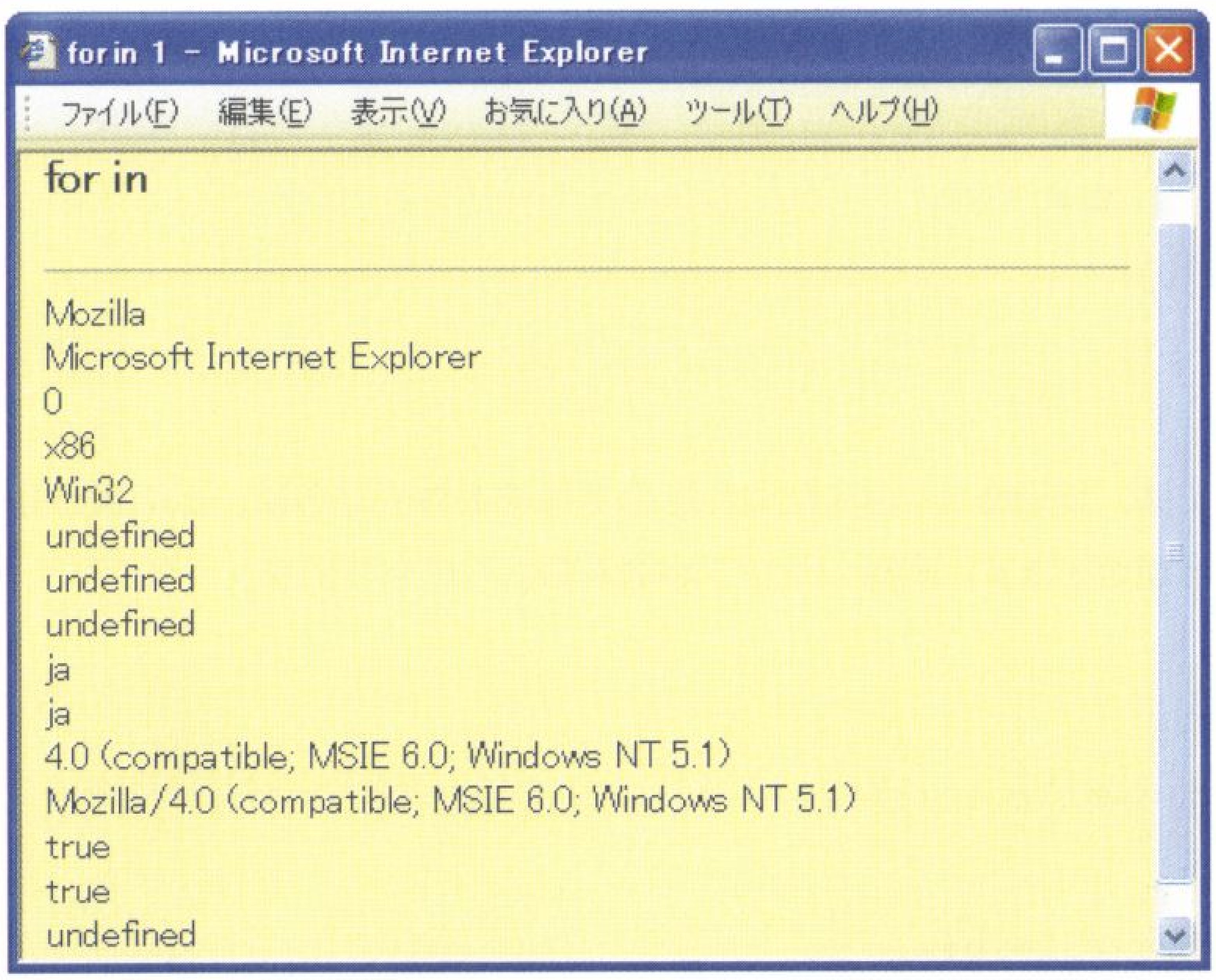
```
for( カウンタ in オブジェクト名 ){
    処理;
}
```

カウンタは0からオブジェクトのプロパティがある間、1ずつ足されて繰り返しを行います。サンプルは次のようになります。

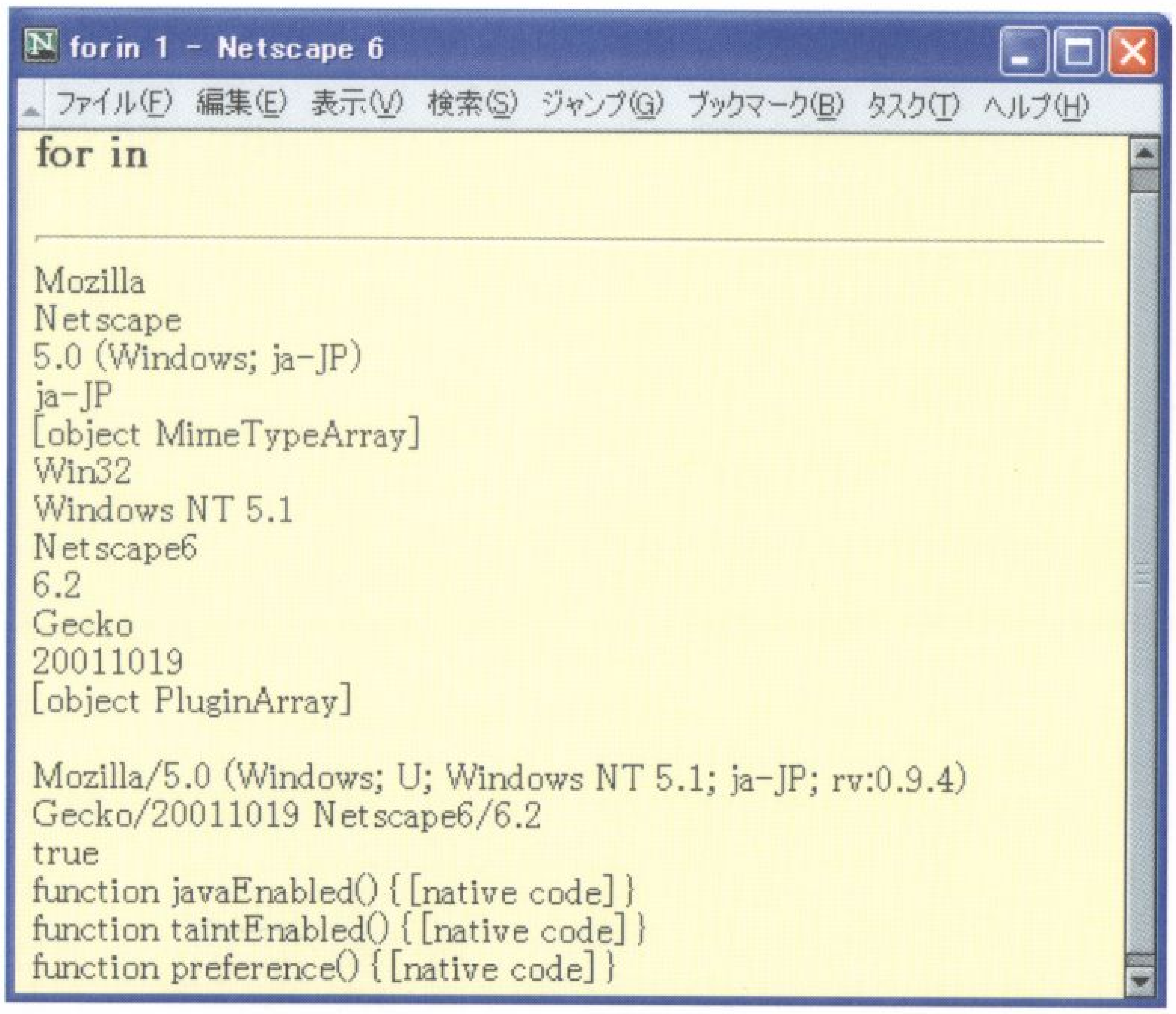
```
<html>
<title>forin 1</title>
<body bgColor="#fff8dc">
<h3>for in</h3>
<hr />
<script language="javascript">
<!--
for(i in navigator){
   document.write(navigator[i]);
   document.write("<br />");
}
//-->
</script>
</body>
</html>
```

上記のスクリプトを実行すると以下のようになります。このスクリプトではnavigatorオブジェクトのプロパティ一覧を表示しています。

▼IE6の表示



▼NN6での表示



これはスクリプト制作時にプロパティの一覧を表示したい場合や、プロパティの番号がわからない場合などに役に立ちます。なおNNでは未定義のプロパティについては表示しないため、表示結果がIEとNNでは異なります。

■while文を使ったループ

繰り返し処理を実現する他のものとして、while文によるループがあります。while文によるループは、ある条件が真である間ループを繰り返すものです。while文の形式は以下のようになります。

```
while( 条件 ){
    処理;
}
```

条件には、論理値(trueまたはfalse)を返す条件式ならば、どのような式でも書くことができます。実際のサンプルは次のようになります。

```
<html>
<title>while 1</title>
<body>
<h3>while 1</h3>
```

```
<hr />
<script language="javascript">
<!--
   var i=0;
   while(i<=7){
      document.write("<font size="+i+">Hello
⇒        JavaScript!</font><br />");
      i++;
   }
//-->
</script>
</body>
</html>
```

このサンプルは「i」が0から7になるまで繰り返されます。「i=8」になった時点でwhile文の条件式が偽となるのでループから抜けます。条件式だけで繰り返しをコントロールしたい場合に役に立ちます。

■break文とcontinue文

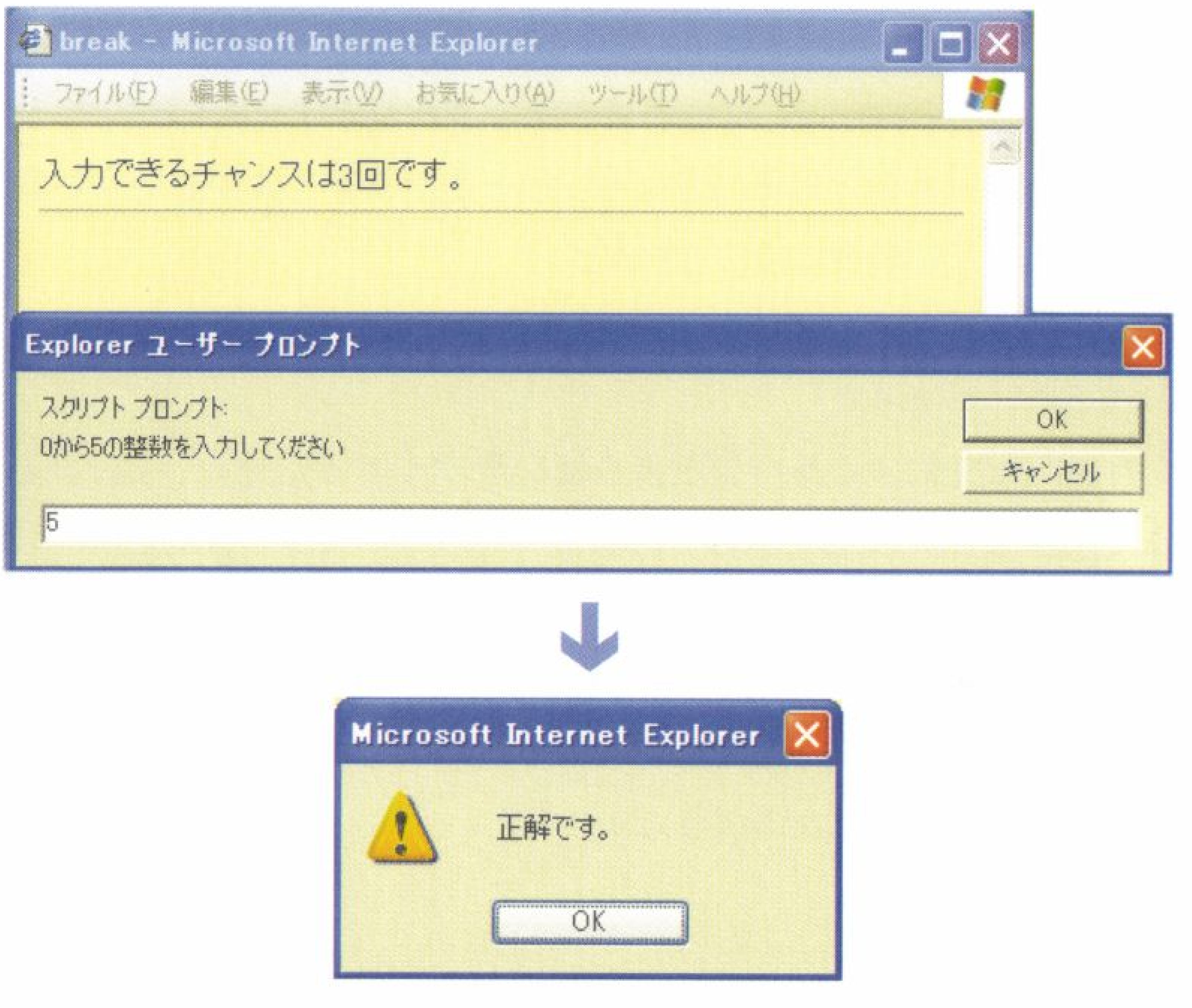
for文とwhile文によるループをより使いやすくするために、JavaScriptはbreak文とcontinue文を用意しています。これらの文は、for文やwhile文の持っている終了条件だけでなく、繰り返し処理中に何らかの条件が成立したらループの動作を変えるといった目的のために使われます。

break文は、ループが終了していなくてもこの命令が実行されるとループから抜け出します。サンプルは次のようになります。

```
<html>
<title>break</title>
<body bgcolor="#fff8dc">
入力できるチャンスは3回です。
<hr />
<script language="javascript">
<!--
   var correct=Math.round(5*Math.random());
   for(i=0;i<3;i++){
```

```
        answer=prompt("0から5の整数を入力してください","");
        if(answer==correct){
            alert("正解です。");
            break;
        }
    }
//-->
</script>
</body>
</html>
```

このスクリプトを実行した結果は、次のように表示されます。



このスクリプトは0から5の整数を生成させて、その値をあてるものです。整数の生成には乱数を使っているため、その値は毎回変わります。答えられる回数はfor文を使って3回に制限し、ここでプロンプトに入力した整数が正答だったときにはif文の条件が真となり、画面にアラートで「正解です。」と表示して、break文によってループを抜けて処理を終了するスクリプトです。

continue文は、ループ中にこの命令を実行させるとそのループ処理を中断し、いったんループ処理の先頭に戻ります。ここで継続処理を行ってから次のループに入ります。break文と異なるのは、ループを抜けてしまわないところです。

2-6

例外処理

プログラムにはバグが付きものです。どんなに気を付けていても、どんなに優秀なプログラマであっても、これは避けられない問題です。さらに、ユーザーはプログラマがどんなに想像をめぐらせても、思い付かない動作をします。こういったバグや、動作により引き起こされる問題を、簡単にプログラム上で処理できるような機構が組み込まれました。それが、「例外処理」です。例外処理は、問題が発生した際に動作します。

JavaScriptで例外処理が扱えるようになったのは最近のことです。JavaScript1.3から例外処理を扱えるようになっています。そのため、これまでは多くのブラウザが対応していなかったため、例外処理を用いることはほとんどありませんでした。

しかし、例外処理を使えば、プログラムの生産性を上げることができます。例外処理を用いないスクリプトでは、検出したいエラーごとにその処理を記述し、さらにそれを処理の近くに記述しなくてはなりませんでした。しかし、例外処理では、例外を検知したときの処理を「catch文」にまとめて記述できるため、正常時の処理と例外処理をきれいに分離できます。これは、プログラム制作時の効率だけでなく、メンテナンスが発生した際のプログラムの理解にとって非常に有効です。そのため、例外処理は可能な限り用いるべきでしょう。

例外処理の実装

例外処理を実装するには以下のような構文を用います。

```
try{
    処理;
}
    catch(例外){
    例外処理;
}
```

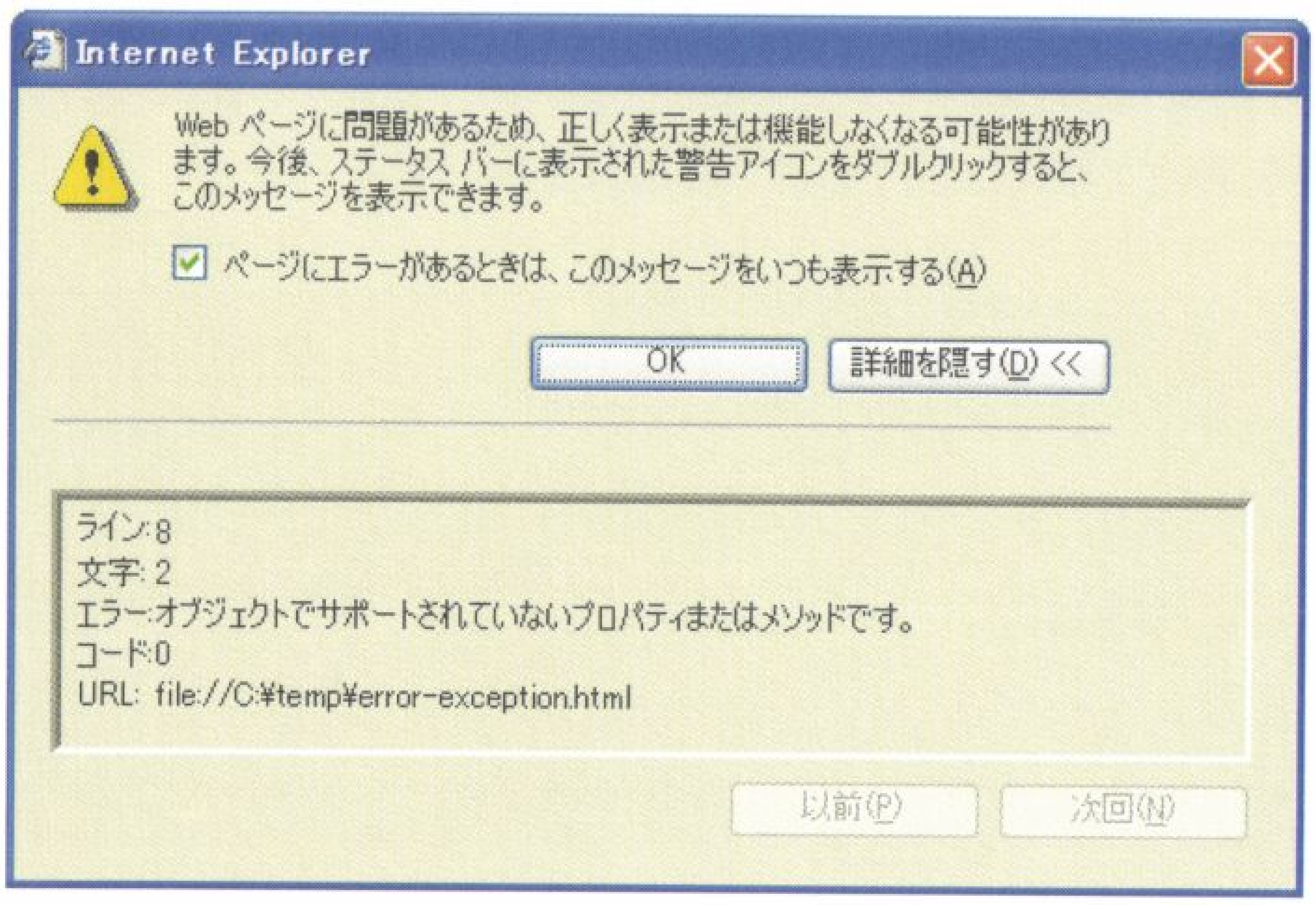
上記のコードでは、try文での後ろの中括弧「{ }」の中に、例外の発生を検知する必要のある処理を記述します。この処理中に問題が発生した場合には、例外が投げられます。この例外を受け取るのが、catch文です。catch文では、try文で

発生した例外を受け取り、引数の部分で処理します。慣例としては「e」を用いることが多いようです。try文には少なくとも1つのcatch文、またはfinally文が必要となります。これがない場合にスクリプトエラーになります。

実際に例外処理のスクリプトを作成してみましょう。まずは例外処理を行っていないスクリプトを見てください。以下のプログラムではdocumentオブジェクトのwriteメソッドの綴りが間違っています。そのため、スクリプトエラーが発生します。

```
<html>
<title>Exception</title>
<body bgcolor="#fff8dc">
<h3>例外処理</h3>
<hr />
<script language="javascript">
<!--
   document.wrte("Error line");
//-->
</script>
</body>
</html>
```

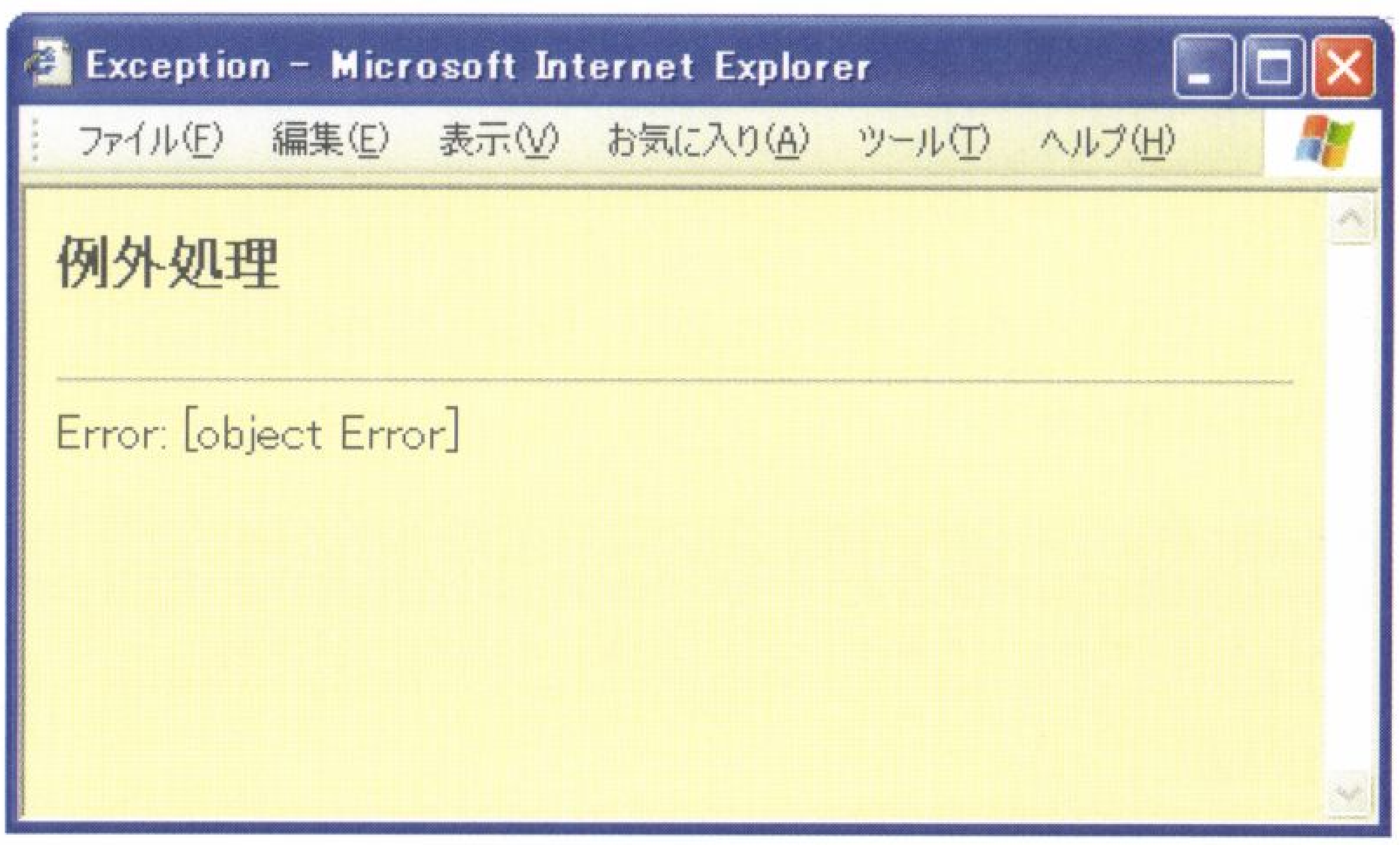
▼IE6で表示されるエラー



それでは、このコードに、例外処理を組み込んでみましょう。スクリプトは次のようになります。

```
<html>
<title>Exception</title>
<body bgcolor="#fff8dc">
<h3>例外処理</h3>
<hr />
<script language="javascript">
<!--
try{
    document.wrte("Error line");
}
catch(e){
    document.write("Error: " + e + "<br />" );
}
//-->
</script>
</body>
</html>
```

例外処理はすでに解説したとおり、try文、catch文により実装されます。問題の起こりそうな部分をtry文で囲み、対応した例外処理をcatch文内に記述します。上記プログラムでは例外の内容を表示しています。実行結果は次のようになります。



例外処理は、入れ子にして記述することもできます。そのため、複雑な処理に対して例外処理を組み込むことができます。

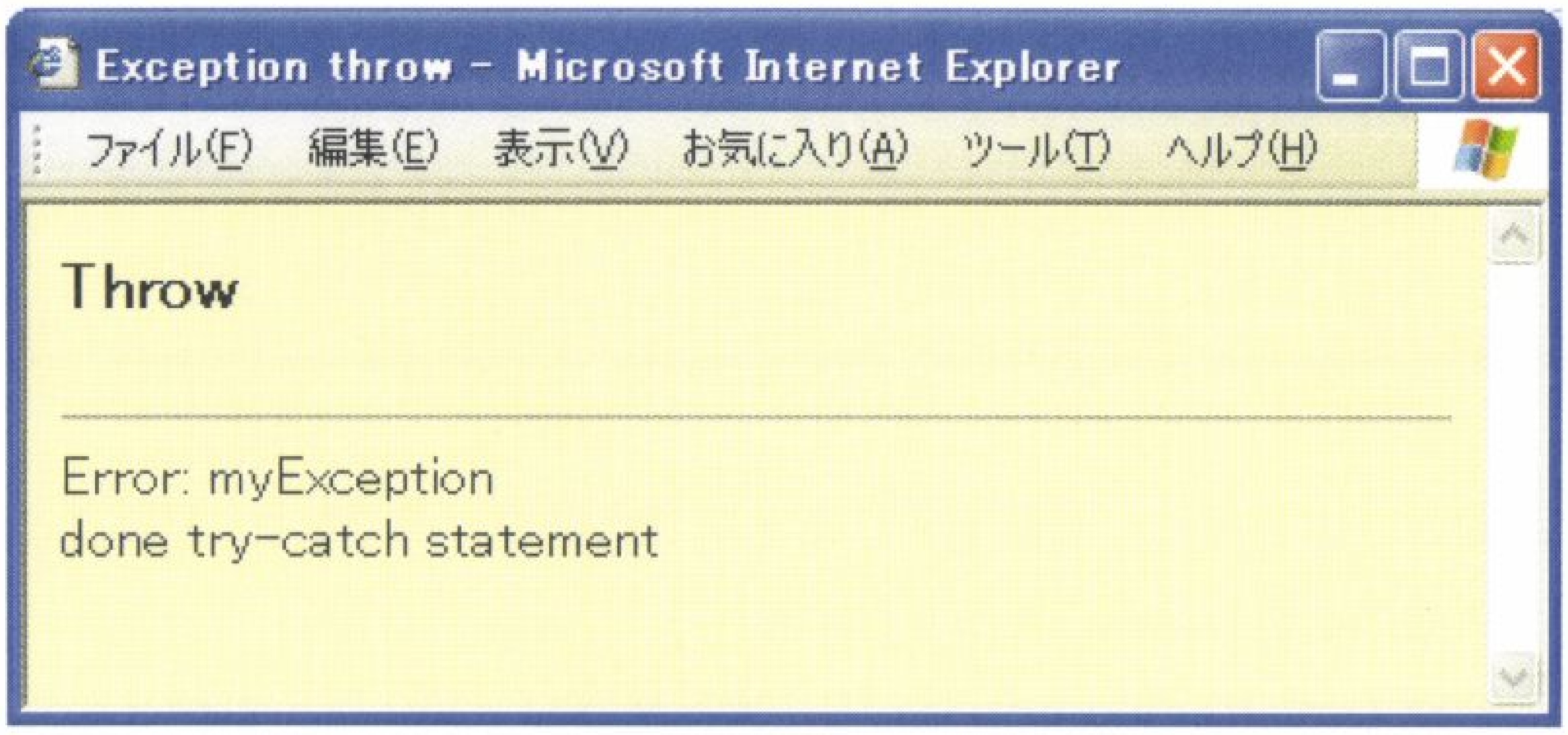
また、例外処理のtry文、catch文の処理を終了した後に行う処理をfinally文

により記述できます。finally文は例外処理が行われる、行われないに関わらず、try文とcatch文の処理が終了した段階で実行されます。

実際のコードは次のようになります。

```
<html>
<title>Exception throw</title>
<body bgcolor="#fff8dc">
<h3>Throw</h3>
<hr />
<script language="javascript">
<!--
   try {
      throw "myException";
   }
   catch (e) {
      document.write("Error: " + e + "<br />");
   }
   finally{
      document.write("done try-catch statement");
   }
//-->
</script>
</body>
</html>
```

上記のコードでは例外処理を終了した後に、finally文内を処理します。実行結果は以下のようになります。



例外を発生する

これまでは、例外を処理する方法を解説してきました。しかし、これだけではまだ思いどおりに例外処理を使いこなすことはできません。そこで必要となるのが、例外を発生させる方法です。スクリプト中で例外を発生させることができれば、エラー処理などが楽になります。

例外を発生させるには、throw文を用います。throwというくらいなので、日本語では「例外を投げる」などともいいます。この発生した例外を受け取るのがcatch文なのです。

throw文の記述は以下のようになります。

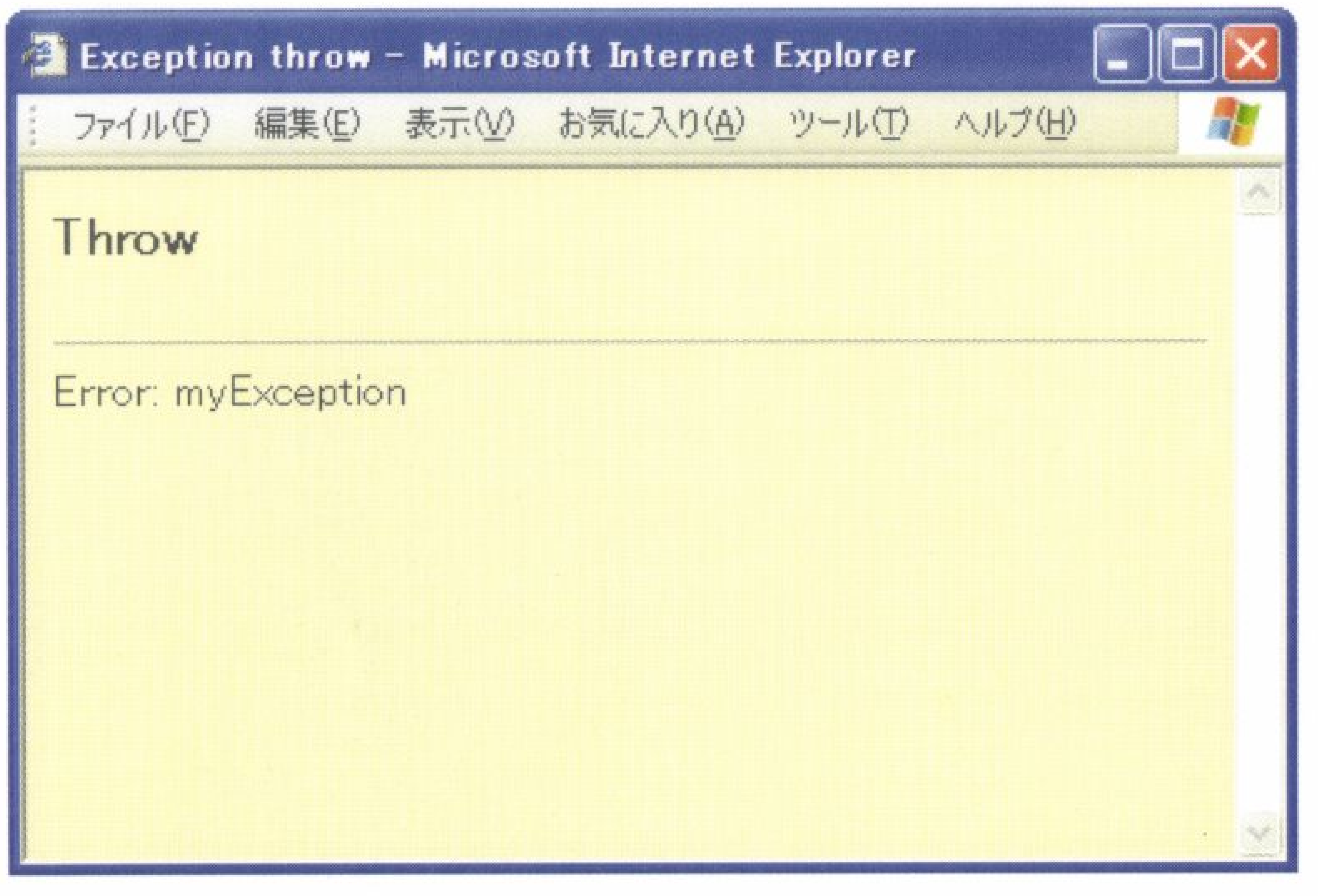
```
try{
    処理;
    throw "式";
}
catch(例外){
    例外処理;
}
```

throw文は引数として式を指定します。throw文の引数には、どのような式でも記述することが可能です。引数に設定された式は、例外を識別するために使われます。

```
<html>
<title>Exception throw</title>
<body>
<h3>Throw</h3>
<hr />
<script language="javascript">
<!--
   try {
      throw "myException";
   }
   catch (e) {
      document.write("Error: " + e );
```

```
}
//-->
</script>
</body>
</html>
```

実行結果は次のようになります。



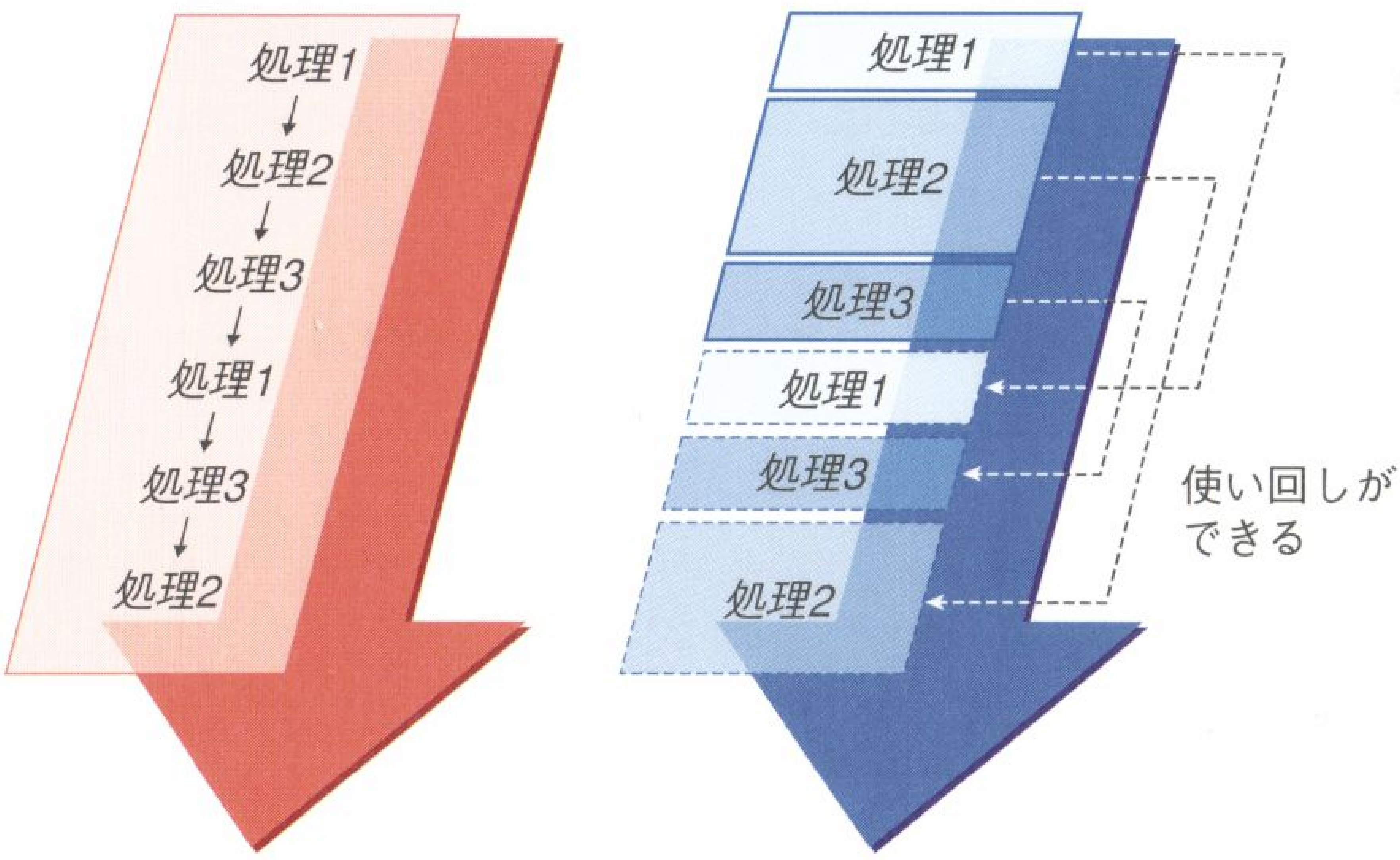
例外処理の注意

ここまで解説したように、例外は実際にプログラムを記述する際には便利なものです。しかし、例外処理には注意する点があります。

まず、最初に触れたように、JavaScriptが例外処理を実装したのが最近のため、実装していないWebブラウザがあります。現在例外処理をサポートしているのは、IE5以降とNN6です。さらに困ったことに、IEとNNでは例外処理の実装が若干異なります。そのため、どちらのWebブラウザかを限定できれば問題なく使うことができますが、両方で使うように記述する場合には、例外処理を分離することぐらいにしか使えないようです。

関数

プログラムやスクリプトを書いていると、同じような処理が何度も出てくることがあります。このようなときに、同じ処理を何度も記述するのはとても面倒だし、効率も悪くなります。そんなときにはどうすればよいのでしょうか？　この問題を解決するのが関数（ファンクション）と呼ばれる機能です。関数はある一連の処理をまとめるためのものです。



たとえば、上の図のようにいくつかの処理が行われる中で、同じ処理が繰り返し出てくる場合、処理の流れに沿って毎回記述していると非常に長いコードになり、コーディングにも時間がかかってしまいます。

そこで、処理単位ごとに関数を作って呼び出せるようにしておくと、何度も同じ処理を記述する必要がなくなり、すっきりとしたスクリプトが作成できます。関数は、メソッドなどと同様に引数を取り、処理をした結果を返します。

関数の定義

関数は「function」を用いて定義します。この定義には、関数の名前と関数に渡す引数リスト、関数内部で行われる処理を記述したスクリプトが必要となります。関数の定義は次のようになります。

```
function 関数名( 引数リスト ){
    処理;
}
```

関数の名前付けのルールは、変数の名前の付け方と同じものです(P.27参照)。具体的には、アルファベット、数字、アンダースコア(_)を使うことができ、大文字と小文字を区別します。また、名前の最初の文字はアルファベット、またはアンダースコアからはじめます。

関数に渡す引数は、関数名の後の括弧内に「,」(カンマ)で区切って並べます。定義された関数が実行されるのは、関数が呼び出されたときのみです。

■引数の渡し方

引数は、実際にはどのようにして使うことができるのでしょうか? サンプルを使って解説しましょう。

```
function putDate( date ){
    document.write(date);
}
```

上記のputDate関数は、dateという引数を渡します。定義された引数はその関数内で自由に使うことができます。関数内で引数を使うと、その中に入っている値は関数に渡したものが参照されることになります。

ここで注意しなくてはならないことは、ある関数に引き渡した引数に関数内で変更を加えたとしても、元の変数の値は変わらないということです。

関数定義部

```
function printName(name){      ← ①関数の読み込み
    name="John";               ← ④nameが変更されている
    document.write(name);
}

name="Mike";                   ← ②変数の定義
printName(name);               ← ③関数の実行。
                                  "John"が表示される

document.write(name);          ← ⑤関数内ではname変更されたが
                                  ここでは"Mike"と表示される
```

※丸数字は実行順序

前ページの図では、printName関数内で、引数で与えられたnameを関数内で変更していますが、関数を呼び出した元の値の内容までは変更されないことになります。

■値を返す関数

関数は処理した結果を返すことができます。このとき返すことのできる値は1つです。返される値を「戻り値」と呼びます。値を返すにはreturn文を使います。形式は戻り値をreturnの後ろに1つ書きます。

```
return 戻り値;
```

これを実際の関数内で使うと、以下のようなコードになります。

```
function Cube( num ){
   cube = num * num * num;
   return cube;
}
```

なお、returnを記述すると、関数の処理はそこで終了します。

関数を実行する

関数の定義は行いました。では実際に実行させるにはどのようにしたらよいのでしょうか？

関数を実行させることを「関数の呼び出し」と呼びます。関数は、値が返ってこない場合は、関数名と引数を書くことで実行されます。書式としては以下のようになります。

```
関数名( 引数 );
```

たとえば、以下のような関数が定義されているとします。

```
function printName( name ){
   document.write( name );
}
```

　このように定義されたprintName関数を実行する場合には、次のように記述します。

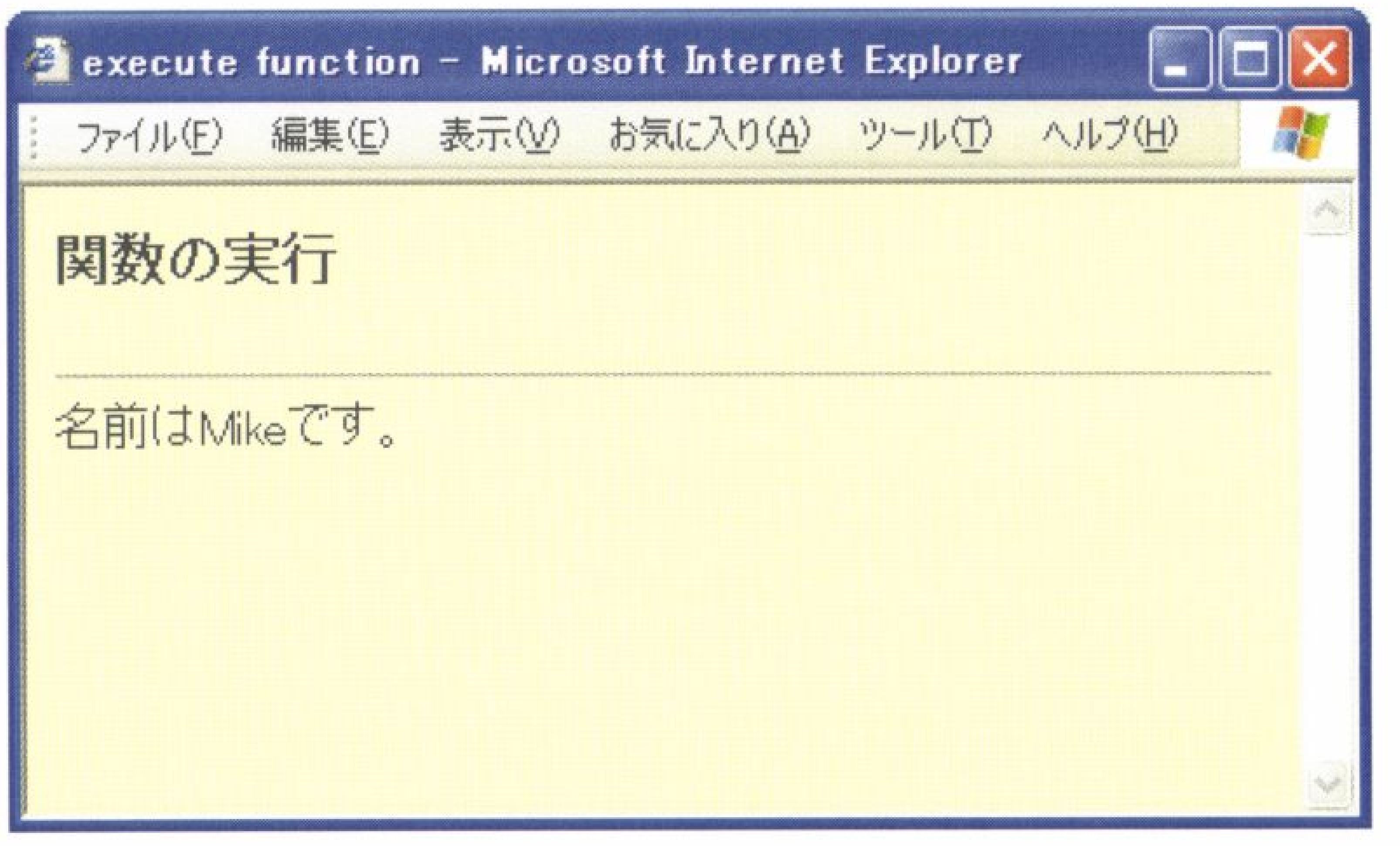
```
printName( name );
```

　関数の呼び出しで注意しなくてはならないのは、関数を実行する前までに、関数の定義が行われていなければならないということです。
　関数を使ったサンプルは以下のようになります。

```
<html>
<title>execute function</title>
<script language="javascript">
<!--
function printName(name){
    document.write("名前は",name,"です。");

}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>関数の実行</h3>
<hr />
<script language="javascript">
<!--
    var name="Mike";        //変数の定義
    printName(name);        //関数の呼び出し
//-->
</script>
</body>
</html>
```

　このスクリプトを実行させると、次のような結果が得られます。



また、値が返ってくる場合は、それを代入するための変数を用意します。したがって関数呼び出しの書式は、以下のようになります。

```
変数 = 関数名( 引数 );
```

たとえば、次のように定義された関数があるとします。

```
function Cube( num ){
   cube = num * num * num;
   return cube;
}
```

この場合には、関数の呼び出しは次のようになります。

```
answer = Cube( num );
```

この「answer」は、返ってくる値を受け取る変数です。実際は好きな名前を付けてかまいません。

■変数の範囲

関数の引数についての注意は、変数がどの範囲で使えるかという問題と同じものです。ここで変数はどの範囲内で使えるかについて解説します。これを、「変数のスコープ」と呼びます。

変数は宣言された場所、はじめて使われた場所によって、どの範囲で使えるか

が決まります。変数が宣言された場所、はじめて使われた場所を大きく分けると、次の2つになります。

●関数内
●それ以外

```
<script language="javascript">
<!--
var Today="2001/9/1";
function PrintBirthday(){
    var myBirthday="1973/9/1";
    document.write(Today);
}
//-->
</script>
```

関数外での宣言（var Today="2001/9/1";）

関数内での宣言（var myBirthday="1973/9/1";）

関数の内部で宣言された変数は、その関数でしか使うことができません。そのため、外部から関数内部で宣言された変数を呼び出しても、定義されていない、または予期せぬ値を参照することになります。

それ以外の場所（<script>～</script>内）で宣言された変数は、どこからでも参照することができます。

```
<script language="javascript">
<!--
var Today="2001/9/1";
function PrintBirthday(){
    var myBirthday="1973/9/1";
    document.write(Today);
}
document.write(myBirthday);
//-->
</script>
```

関数外で定義された変数は参照可能

関数内で定義された変数は参照できない

このため、さまざまな関数内で共通に使いたい変数（たとえば日付などを格納するもの）は関数の外で宣言しておきましょう。また逆に、関数内でしか使わない変数は関数の中で宣言しておくのがよいでしょう。

2-8 オブジェクトモデル

JavaScriptのスクリプトプログラミングの中で、非常に重要な役割を果たすのが「オブジェクト」です。いきなりオブジェクトといってしまうと、とても難しいもののように聞こえますが、一度理解してしまえばとても簡単です。また、このオブジェクトの定義は、いわゆるオブジェクト指向言語の一部だけ抜き出したようなものなので、Java言語を代表とするオブジェクト指向言語を少しでもわかっていると容易に理解できます。

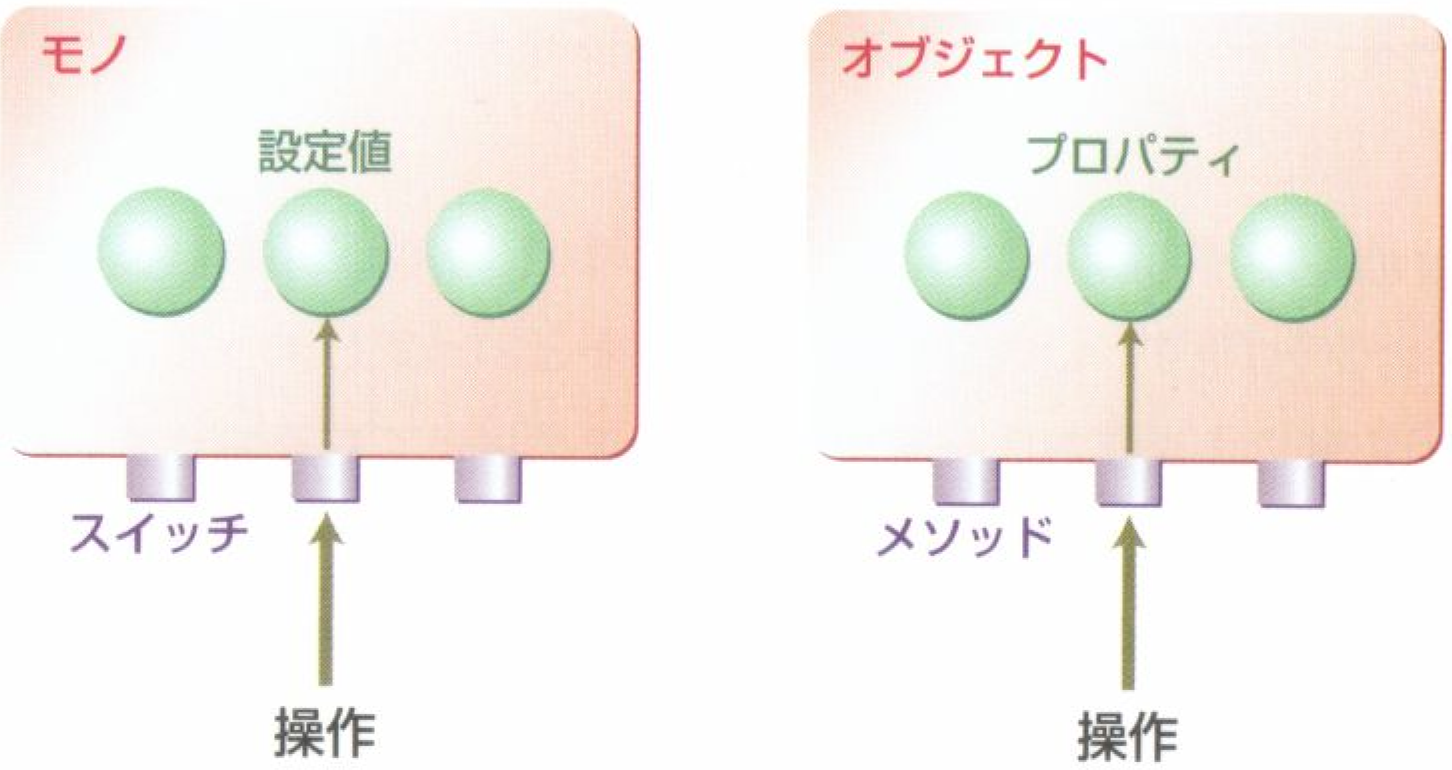
オブジェクトとは

さてオブジェクトとは、いったいどのようなものなのでしょうか？　日常生活をしていると、いわゆる「モノ」をよく使います。たとえばテレビ、電話、カメラ、炊飯器、電子レンジ………。これらはどれも「モノ」です。これらの「モノ」の仕組みは複雑ですが、実際にユーザーが使うときには簡単に使うことができます。それは、使うのに必要な操作が限られているからです。

たとえば、炊飯器を考えたとき、お米を炊くために自分で温度や時間などを設定し、さらにはその構造や電気の流れ方まで指定する必要はありません（古い炊飯器なら、もしかしたら温度や時間を設定しないといけないのかな？）。普段、そんなことを考えなくても炊飯器を使えているのは、炊飯器を使うための操作が限られているからです。「炊く」、「保温する」………、といった操作は、その操作に対応するスイッチを押すだけで、ほとんど自動で目的の動作（お米を炊くなど）を得られます。このような「モノ」の概念をプログラミングの世界に取り入れたのがオブジェクト指向です。

「モノ」では、「モノ」の持っている各種の値を設定し、操作することができます。具体的にどのような値が設定できるかを挙げてみると、時間、温度、強さ……、といったものになります。また、それぞれの値を設定するにはスイッチが必要となります。「オブジェクト」を、この「モノ」と対比して考えると、次のような対応関係になります。

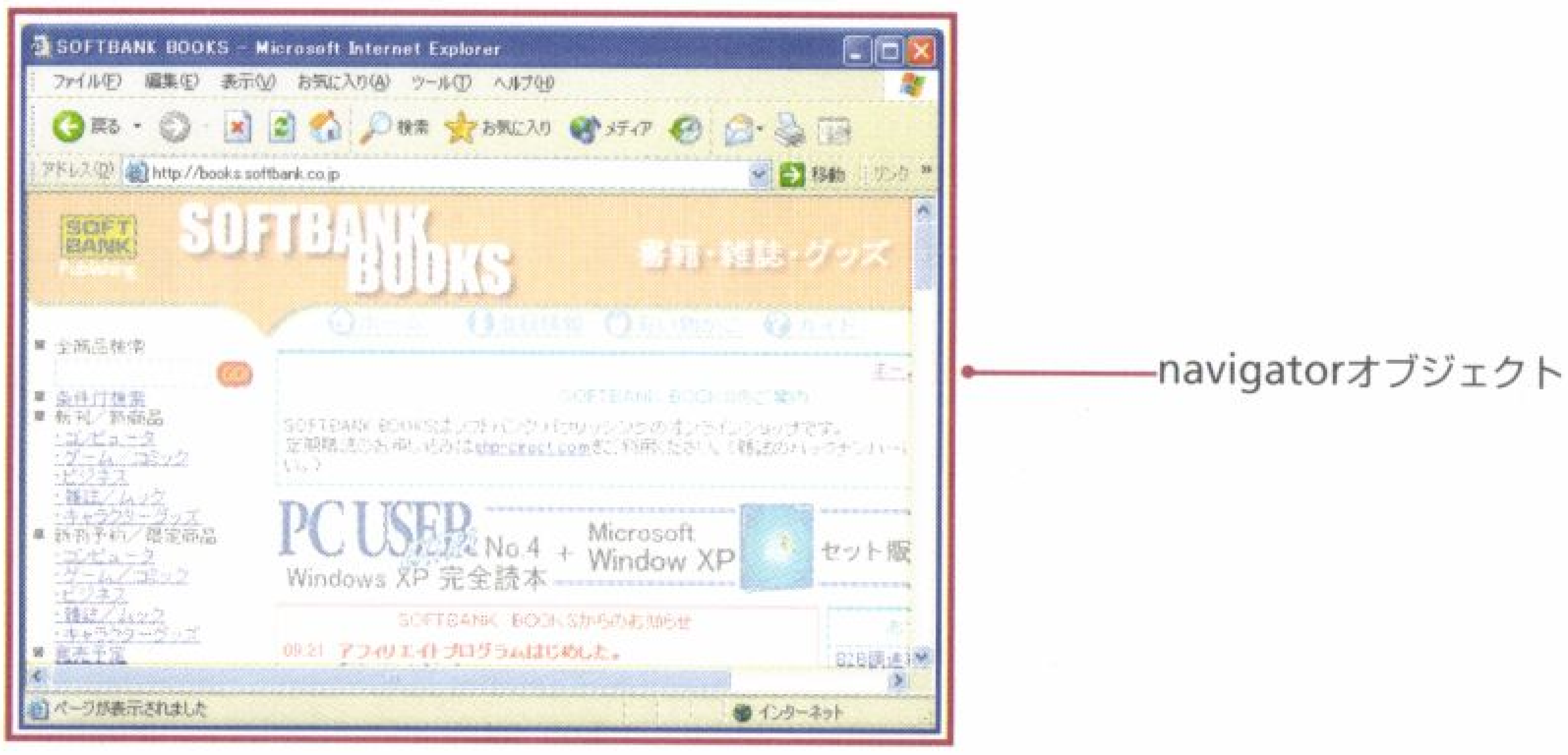
- 「オブジェクト」　：　「モノ」
- 「メソッド」　：　「スイッチ」
- 「プロパティ」　：　「設定値」

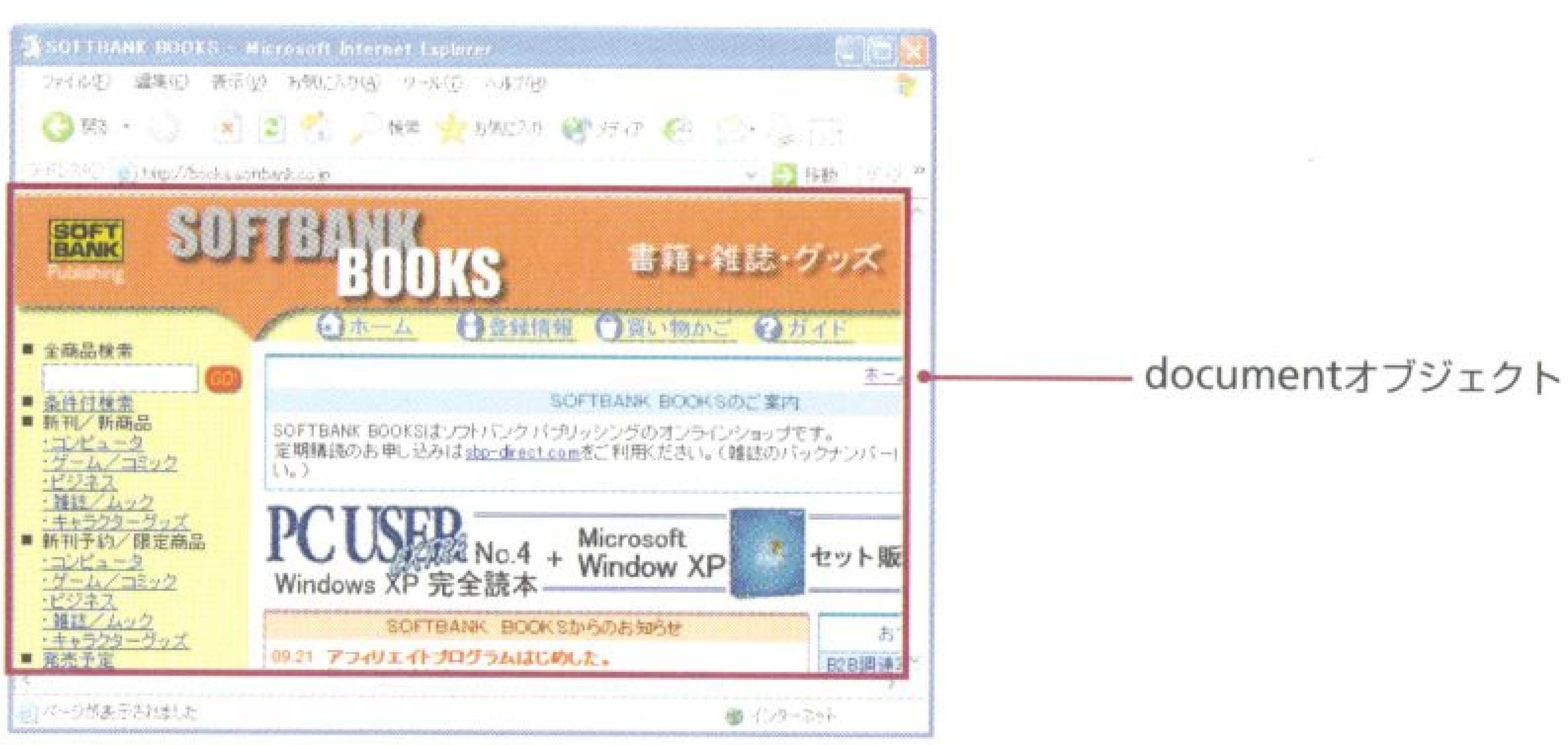


このように考えると、オブジェクトを比較的簡単に理解することができるでしょう。結局のところ、プログラミングの世界でもオブジェクトを動作させるにはスイッチに相当するメソッドというものを使って操作することになります。

ここで、JavaScriptで扱うオブジェクトの例として、わかりやすいものを挙げると、次のようなものがあります。

- navigatorオブジェクト　：　Webブラウザの本体
- documentオブジェクト　：　Webページが表示される部分





ただし、JavaScriptは、Java言語やC++言語などの純粋なオブジェクト指向から見ると、継承にあたる機能がないなど、オブジェクト指向とは言い切れない部分があります。

ビルトインオブジェクトモデル

オブジェクトには、最初からJavaScriptに組み込まれているビルトイン(組み込み)オブジェクトがあります。たとえば、先ほどのnavigatorオブジェクトやdocumentオブジェクト、Formオブジェクト、Dateオブジェクト(日付オブジェクト)、Mathオブジェクト(算術オブジェクト)などがビルトインオブジェクトです。このうちnavigatorオブジェクト、documentオブジェクト、Formオブジェクトは目に見えるオブジェクト、Dateオブジェクト、Mathオブジェクトは目に見えないオブジェクトです。

このようなビルトインオブジェクトがあるのは、Webページ制作者が基本的な部分のプログラムを書かなくても、Webページ上で高度な処理ができるようにするためです。もしも基本的な部分がビルトインオブジェクトとして組み込まれていないと、Webページ上のマウスカーソルのX座標とY座標を調べるプログラムや、ログの値や平方根の値などを求めるための計算用プログラムまで作成しなくてはなりません。これではWebページの制作はとても苦痛なものとなります。こうした問題が起こらないようにするために、Webブラウザ上で使うものをオブジェクトとして実装し、メソッドを用いて利用できるようにしているのです。

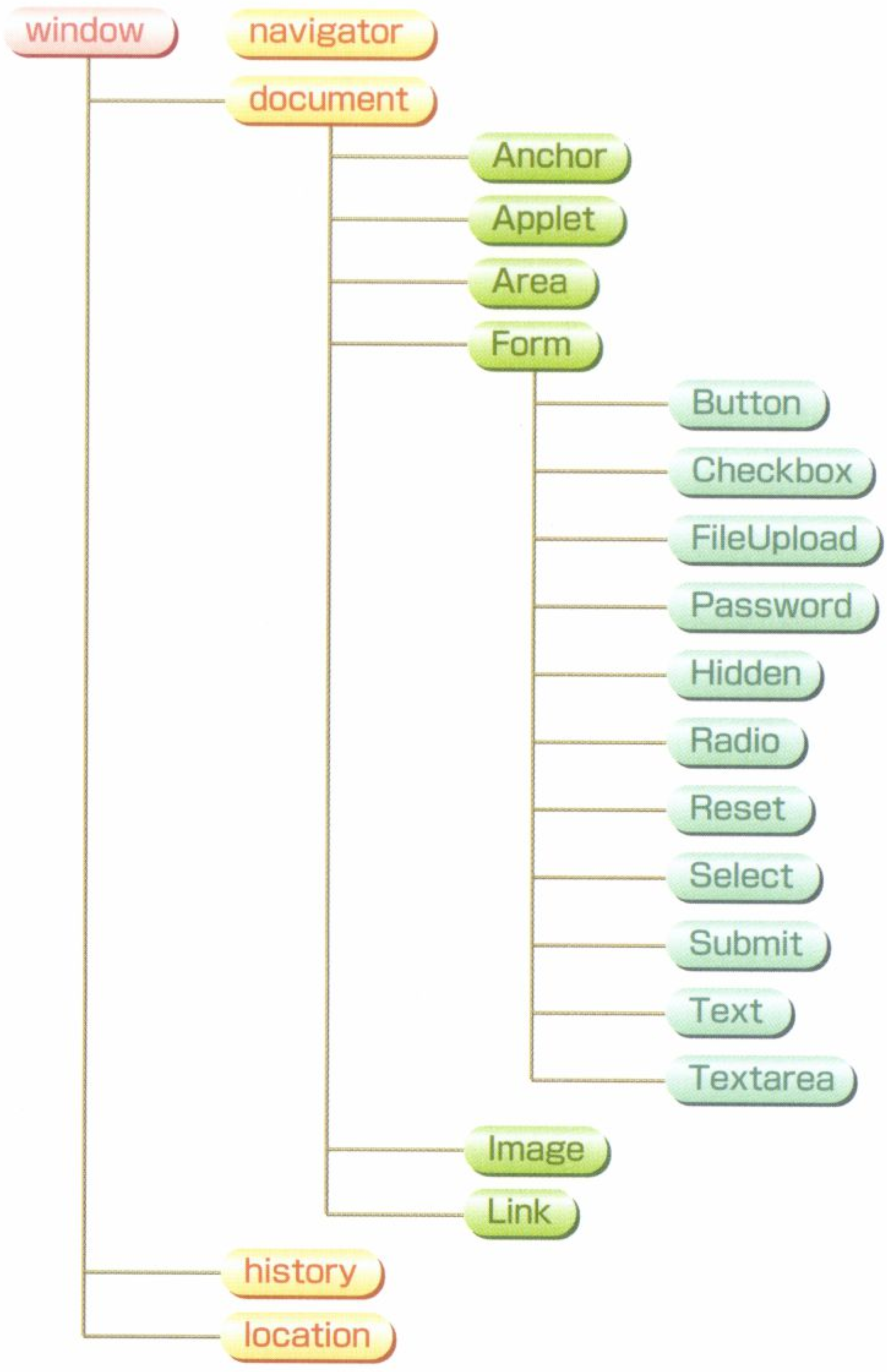
●ビルトインオブジェクト一覧

Anchor	anchors array	Applet	applets array
Area	arguments array	Array	Button
Checkbox	Data	document	elements array
embeds array	FileUpload	Form	forms array
Frame	frames array	Function	Hidden
History	history array	Image	images array
Link	links array	location	Math
MimeType	mimeTypes array	navigator	Number
Option	options array	options array	element
Password	Plugin	plugins array	Radio
Reset	Select	String	Submit
Text	Textarea	window	

■オブジェクトの配置

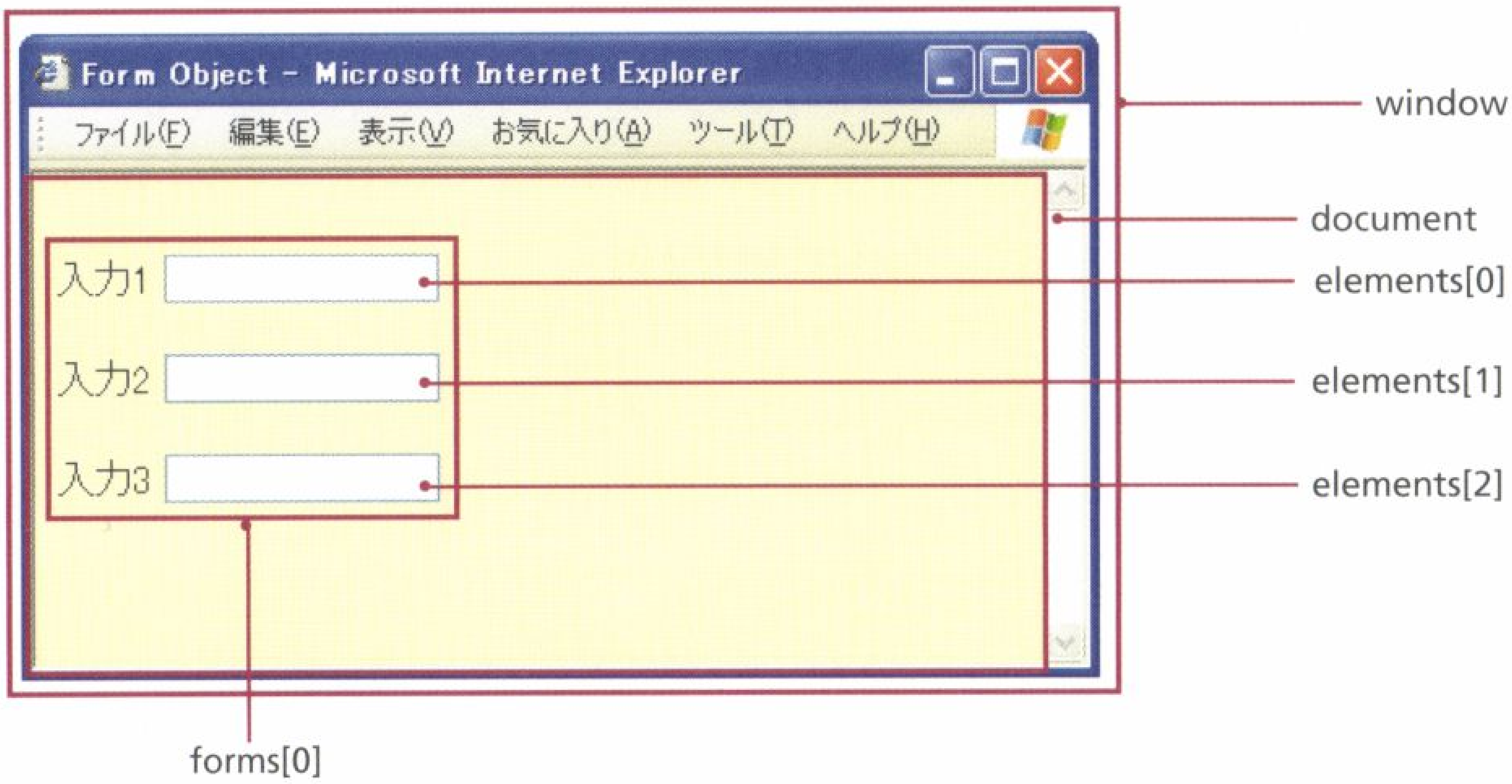
JavaScriptの大半のビルトインオブジェクトは、windowオブジェクトをトップにした階層構造に組み込まれています。この階層をオブジェクト階層と呼びます。また、この階層構造自体をオブジェクトモデルと呼ぶこともあります。

▼オブジェクトモデル



このような階層構造になっているのは、オブジェクトのアクセスを容易にするためです(ただ、NNとIEのオブジェクトモデルは若干異なります)。

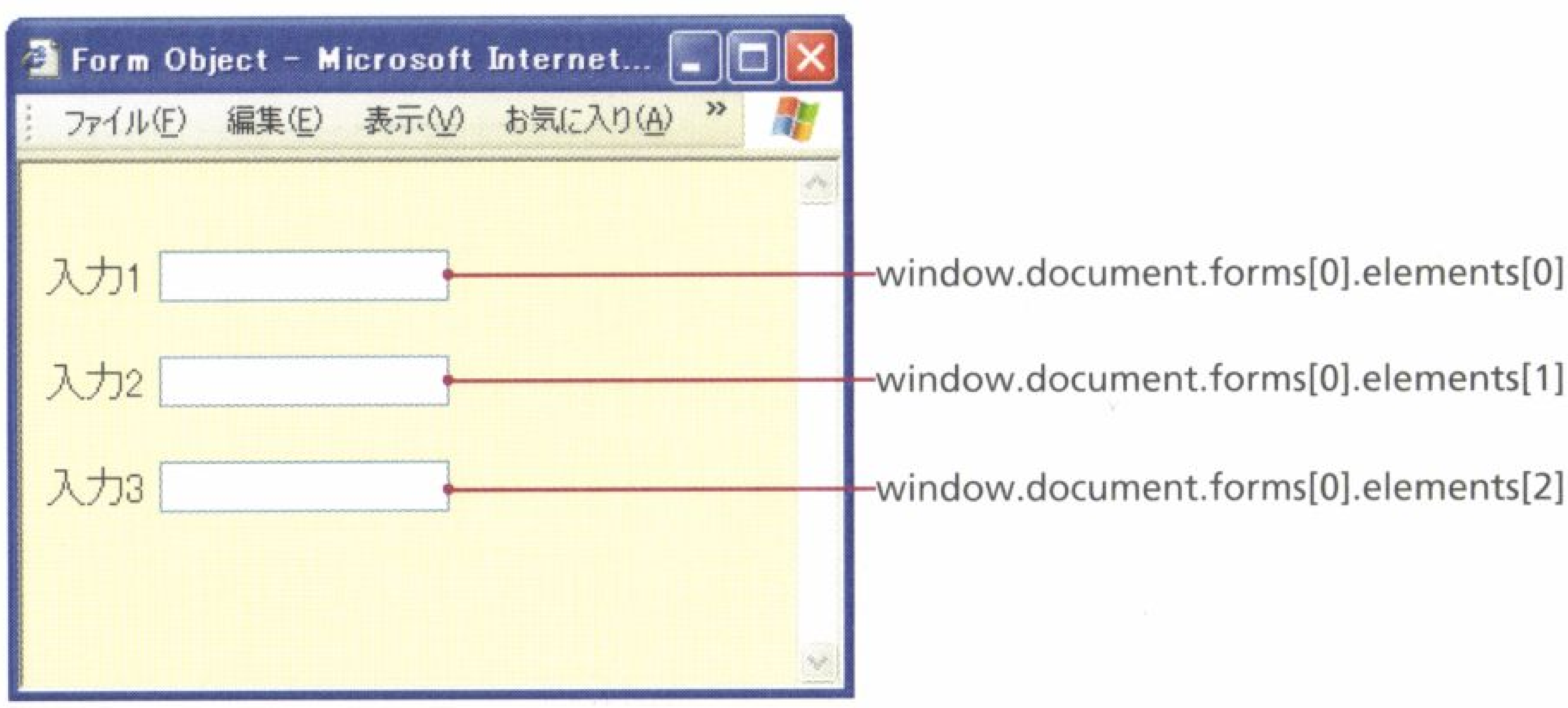
たとえば、インプットフォームが配置されているようなWebページを考えてみましょう。次ページの図は、このときのオブジェクトの範囲と名前を対応させたものです。



このようにWebブラウザのそれぞれのオブジェクトには対応する名前が付いています。なお、同じ要素が並んだ場合は配列として扱われ、[ ]内の数字で区別します。[ ]内の数字は0からはじまります。

なぜこのような構造になっているかというと、Webページ内が構造を持たないとすると、それぞれのフォームなどのオブジェクトを管理するために、各オブジェクトに識別できるための名前を付ける必要があります。これは一見簡単に見えますが、プログラムでコントロールしたい場合には面倒になります。そのため、WebブラウザはWebページを階層構造により管理しています。なお、それぞれのオブジェクトに名前を付けることもできます。

オブジェクトの階層構造にしたがって、それぞれのオブジェクトへアクセスする名前を考えてみると、次の図のようになります。



それでは、主要なオブジェクトを取り上げて、解説していきましょう。

KNOWLEDGE

■windowオブジェクト

windowオブジェクトは、すべてのオブジェクトの基となるものです。このオブジェクトが扱うものは、ウィンドウの操作や、メニューバー、ツールバー、アラートといったWebブラウザ全体の動作です。

windowオブジェクトはすべてのオブジェクトの親なので、参照するときに記述を省略してもかまいません。たとえば、アラート(警告のダイアログボックス)を呼び出す場合、本来なら、次のように記述しなくてはなりません。

```
window.alert();
```

しかし、windowオブジェクトは、必ず存在するものなので、オブジェクトを指定するのに、windowオブジェクトを省略して記述することができます。そのため、アラートを呼び出すのには次のように記述することもできます。

```
alert();
```

●windowオブジェクトの主なプロパティ

closed	defaultStatus	frames	length	name	opener
parent	self	status	top	window	

●windowオブジェクトの主なメソッド

Alert	blur	clearTimeout	close	confirm	focus
open	prompt	scroll	setTimeout		

■navigatorオブジェクト

navigatorオブジェクトは、Webブラウザについての情報を持っているオブジェクトです。このオブジェクトを使うことによりユーザーが使っているWebブラウザやOSなどの環境について調べることができます。

●navigatorオブジェクトの主なプロパティ

appCodeName	appName	appVersion	mimeTypes
plugins	userAgent		

●navigatorオブジェクトの主なメソッド

javaEnabled

■documentオブジェクト

documentオブジェクトは、「Webページを表示する部分」を表すオブジェクトです。documentオブジェクトには、アンカー、フォーム、リンク、タイトル、カラー、場所など、ページの要素に関するさまざまな情報が含まれており、多くの要素オブジェクトを操作するメソッドやプロパティを提供しています。実質的にはdocumentオブジェクトは<body>～</body>で囲まれる部分に対応する、と考えることができます。

この範囲を操作する場合は、ほとんどdocumentオブジェクト以下のメソッドやプロパティ、オブジェクトを扱えばよいことになります。

●documentオブジェクトの主なプロパティ

alinkColor	anchors	applets	bgColor	cookie
domain	embeds	fgColor	forms	images
lastModified	linkColor	links	referer	title
URL	vlinkColor			

●documentオブジェクトの主なメソッド

close	open	write	writeln

documentオブジェクトは、formオブジェクトやlinkオブジェクトなどを子として持っています。

■locationオブジェクト

locationオブジェクトは、windowオブジェクトの下に配置されているオブジェクトです。このオブジェクトは、場所（URL）に関しての処理を行うときに使います。URLを変更したり、現在表示されているページのアドレスについての情報を取り出したりすることができます。

●locationオブジェクトの主なプロパティ

hash	host	hostname	href	pathname
port	protocol	search		

●locationオブジェクトの主なメソッド

reload	replace

■historyオブジェクト

historyオブジェクトは、Webブラウザの持っている履歴に関しての処理を行うときに使います。このオブジェクトを用いることにより、1つ前のページへ戻るといった動作をさせることができます。

●historyオブジェクトの主なプロパティ

current	length	next	previous

●historyオブジェクトの主なメソッド

back	forward	go

■オブジェクトの生成

オブジェクトを使うには、オブジェクトを生成する必要があります。オブジェクトによっては1つのWebページ内で複数使われる可能性があります。そのため、事前にオブジェクトを用意しておくとすると、Webブラウザはいくつのオブジェクトを生成しておく必要があるかわかりません。そのため、オブジェクトを使うためには、明示的にオブジェクトを生成する必要があります。このとき生成されたオブジェクトを「オブジェクトの実体」と呼ぶこともあります。

オブジェクトは生成するときには名前が必要です。この名前はWebページ内でオブジェクトを一意に区別するために用いられるので、1つのWebページ内で同じ名前を使うことはできません。

オブジェクトの生成には「new」を使います。書式は以下のようになります。

```
ページ内で認識するオブジェクト名 = new オブジェクト名();
```

上記のように記述すると「オブジェクト名」と同じ機能を持ったオブジェクトが生成されます。

たとえば、次のような記述があるとします。

```
myDate = new Date();
```

これだと、Dateオブジェクトを基にした「myDate」オブジェクトが生成されます。この生成はWebページ内で複数使用される可能性のあるオブジェクトにだけ行わなければなりません。逆にWebページ内に1つしかないようなnavigator、historyなどは、オブジェクトの生成を行わなくても使用することができます。

■with文

JavaScriptではスクリプトの中でオブジェクトの階層を記述します。これはページ内で論理的にオブジェクトを扱うために便利なのですが、入力は面倒なものです。同じような階層構造を記述する場合には、その都度記述しなくてはなりません。そういった場合には、with文を使うことで共通するオブジェクトの記述を省略できます。書式は以下のようになります。

```
with(オブジェクト名){
    処理;
}
```

たとえば、文字列を画面に表示するような場合、次のように記述します。

```
document.write("第1行目");
document.write("第2行目");
document.write("第3行目");
```

この記述は、documentオブジェクトを繰り返して記述しています。この部分をwith文を用いて以下のように書き換えることができます。

```
with(document){
    write("第1行目");
    write("第2行目");
    write("第3行目");
}
```

with文は、入れ子に指定(with文の中でwith文を指定)することも可能です。

メソッド

メソッドとは、オブジェクトを操作するためのスイッチです。オブジェクトは、基本的にメソッドにより操作されます。オブジェクトには、それぞれを操作するためのメソッドが用意されています。JavaScriptでは、このメソッドを使いオブジェクトを操作することでプログラミングを行います。

JavaScriptでのメソッドの記述は、ちょうど関数の呼び出しと同じような形を

しています。あるオブジェクトのメソッドを操作するには、「どのオブジェクト」に対して「どのような操作」をするかを指定します。この関係を記述するのにJavaScriptでは「.」(ピリオド)を用います。

メソッドの記述の書式は以下のようになります。

```
オブジェクト名.メソッド名();
```

次の利用方法の例は、Webページが表示される部分に該当するdocumentオブジェクトに対して、writeメソッドを使うことで、Webページに文字列を表示することができます。

```
document.write(" Hello! JavaScript ");
```

プロパティ

プロパティとは、オブジェクト内であらかじめさまざまな値を保持しているものです。オブジェクト内の固有の変数ということもできます。

JavaScriptでは、プロパティはビルトインオブジェクトがもともと持つものとして実装されています。プロパティに値を設定することにより、オブジェクトの設定値を変更することができます。プロパティの値として設定されているのは、背景色、テキストの色、Webブラウザの各種情報(ブラウザ名、バージョン情報など)などです。

プロパティの使い方は、メソッドと同様にオブジェクトを指定してプロパティを指定し、「どのオブジェクト」の「どの値」と指定します。メソッドと異なるのは、単なる値なのでプロパティ名の後ろに「( )」を付けないことです。

プロパティの書式は以下のようになります。

```
オブジェクト名.プロパティ名
```

実際に、プロパティへのアクセスを行う記述を行うと、次のようになります。

```
document.bgColor;
navigator.appName;
this.value;
```

プロパティは、値を設定したり参照したりすることができます(ただし、プロパティによっては値が設定できないものもあります)。プロパティに値を設定するには、プロパティに値を代入することにより行います。書式は以下のようになります。

```
オブジェクト名.プロパティ名 = 値;
```

この記述からわかるように、代入演算子を用いて、プロパティに値をセットします。記述例は、次のようになります。

```
document.bgColor ="#ff0000";
document.forms[0].elements[0].value = 123;
```

またプロパティを参照するには、プロパティを指定することで行います。プロパティの値の参照は、代入演算子で読み出すこともできますし、メソッドの引数の部分でも行うことができます。

```
変数 = オブジェクト名.プロパティ名;
メソッド( オブジェクト名.プロパティ名 )
```

記述例としては、次のようなものになります。

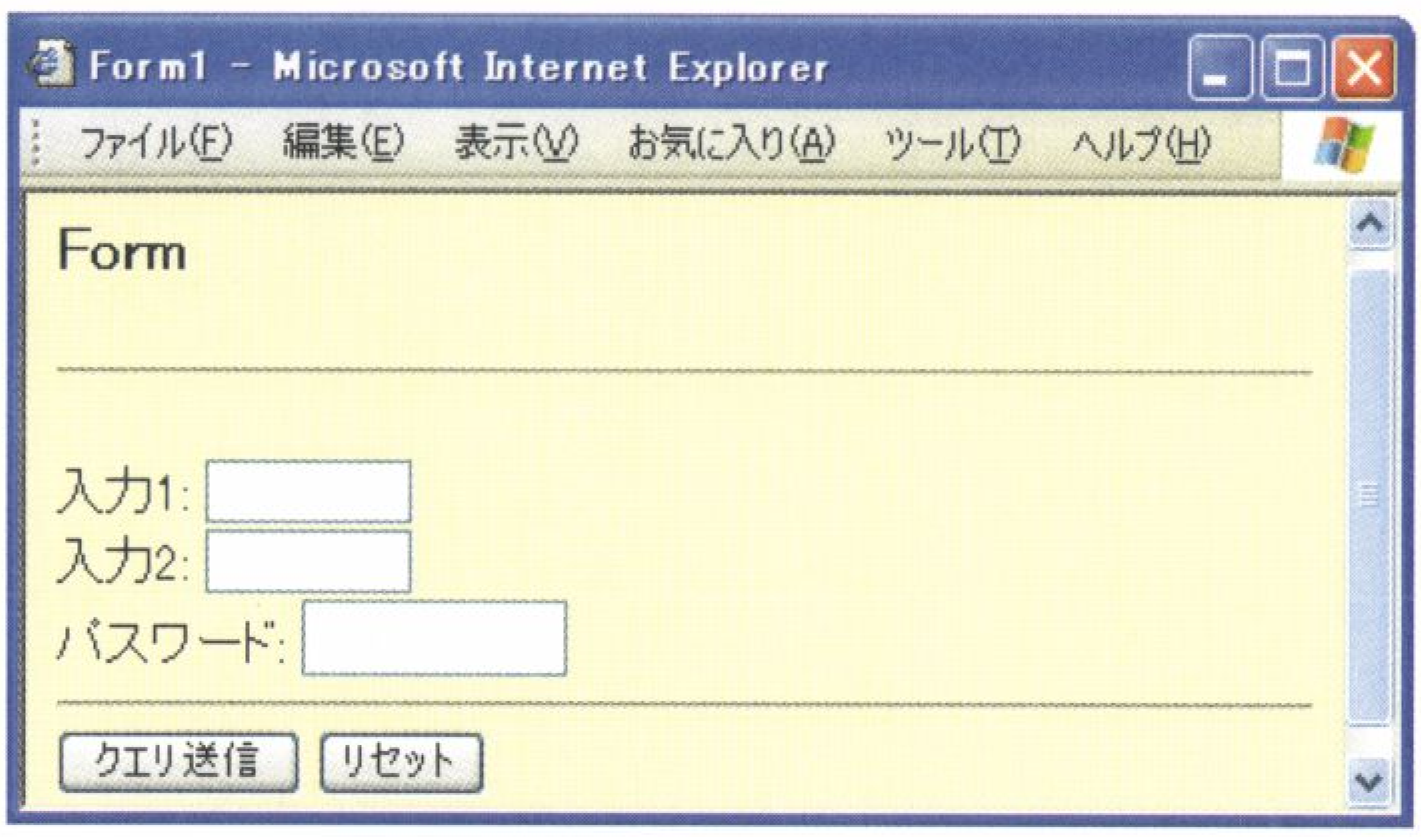
```
a = document.forms[0].elements[0].value;
document.write( navigator.appName );
```

2-9 フォーム、フレームを扱う

フォームとは

フォームはWWWに対して非常に多くの影響を与えてきました。特にCGIと連携することにより、さまざまなWebアプリケーションを実現するための部品として使われています。フォームのサンプルを挙げると、次のような記述と表示になります。

```
<form method="post"
    action="http://www.＊＊＊.jp/cgi-bin/check">
    入力1: <input type="text" name="input1" size="10"/>
        <br />
    入力2: <input type="text" name="input2" size="10"/>
        <br />
    パスワード: <input type="password" name="password1"
        size="10" />
    <hr />
    <input type="submit" value="クエリ送信" />
    <input type= "reset" value="リセット" />
</form>
```



このサンプルコードでは、上の図のように入力フォームが3つ表示されます。これに値を入力し、クエリ送信をクリックすると「http://www.＊＊＊.jp/cgi-bin/check」で指定したWebサーバー上のCGIプログラムを実行します。

しかし、「入力された値が正しい形式かどうかチェックしたい」などといった場合にはどうしたらよいのでしょうか？　入力された値をWebサーバーへ送り、そこで値を検証して結果を返すということもできますが、レスポンスはかなり遅いものになるでしょう。このような状況で威力を発揮するのがJavaScriptです。JavaScriptでは、クライアントサイドで入力された値をチェックしたり、入力された値に応じた反応を返したりといったことを簡単に行えます。

ここでは、JavaScriptを使ってインタラクティブな反応ができる、フォーム制作のための基礎知識を解説します。

フォームを作る

最初に、簡単にHTMLのフォームを思い出してみましょう。フォームで必要となる要素は<form>タグです。書式は以下のようになります。

```
<form method="post | get" action="プログラムへのパス">
    フォーム要素のオブジェクト
</form>
```

method属性は、フォーム要素のオブジェクトに入力された値をどのような方法で渡すかというものです。またaction属性には、submit（提出）されたときにどのプログラムに処理を引き渡すかということを指定しておきます。

主なフォーム要素のオブジェクトには、以下の表のものがあります。

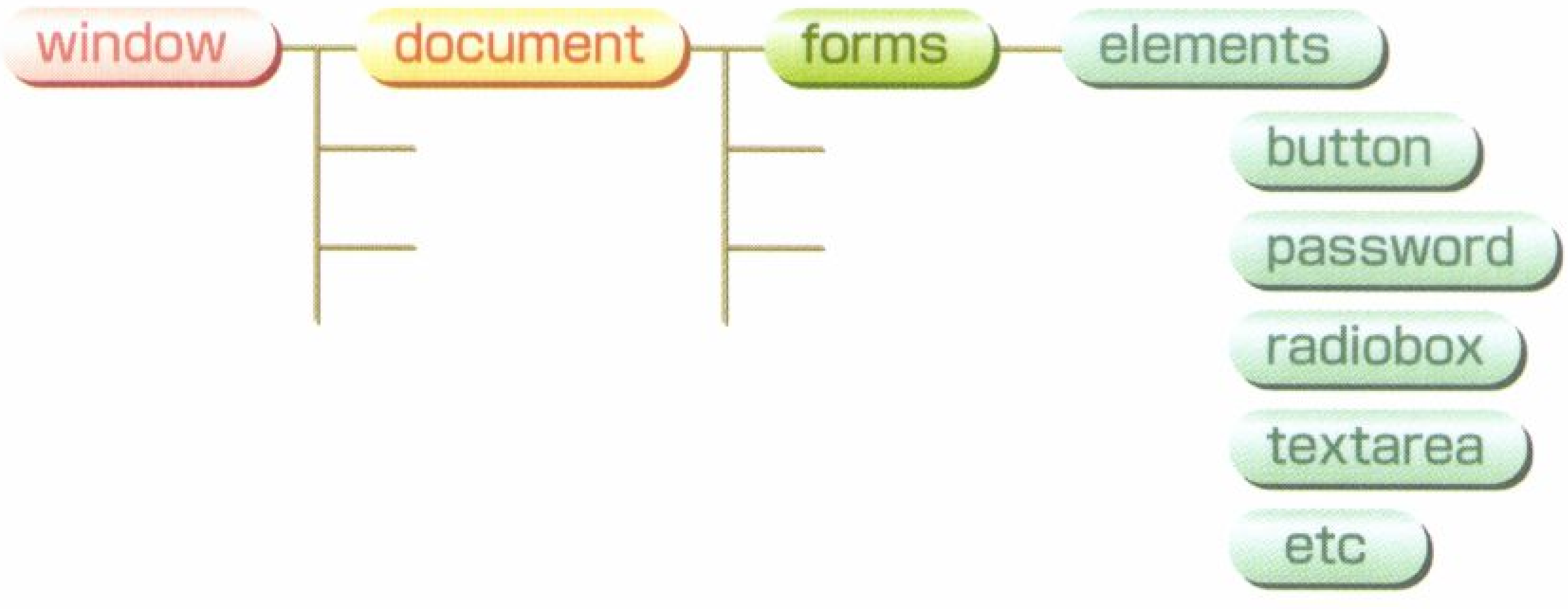
●主なフォーム要素のオブジェクト

オブジェクト	メソッド	イベントハンドラ
button	click()	onClick
checkbox	click()	onClick
hidden		
password	focus()、blur()、select()	onFocus、onBlur、onSelect
radio	click()	onClick
reset	click()	onClick
select		onBlur、onFocus、onChange
submit	click()	onClick
text	focus()、blur()、select()	onFocus、onBlur、onSelect
textarea	focus()、blur()、select()	onBlur、onFocus、onChange、onSelect

フォームを扱う

フォームをJavaScriptで扱うにはどうしたらよいのでしょうか？　ここで重要となるのがオブジェクト階層です。フォームの階層構造はwindowオブジェクトから見てくると次のようになります。

▼フォームオブジェクト階層



elementsはフォーム要素のオブジェクトを表しています。formsオブジェクトとelementsオブジェクトは、Webページ内に複数ある可能性があります。上図のようにフォームはさまざまなオブジェクトの子です。JavaScriptでこのフォームを参照するには、階層構造を意識して指定します。

JavaScriptではフォーム全体、フォームの各要素を示すキーワードは以下のようになります。

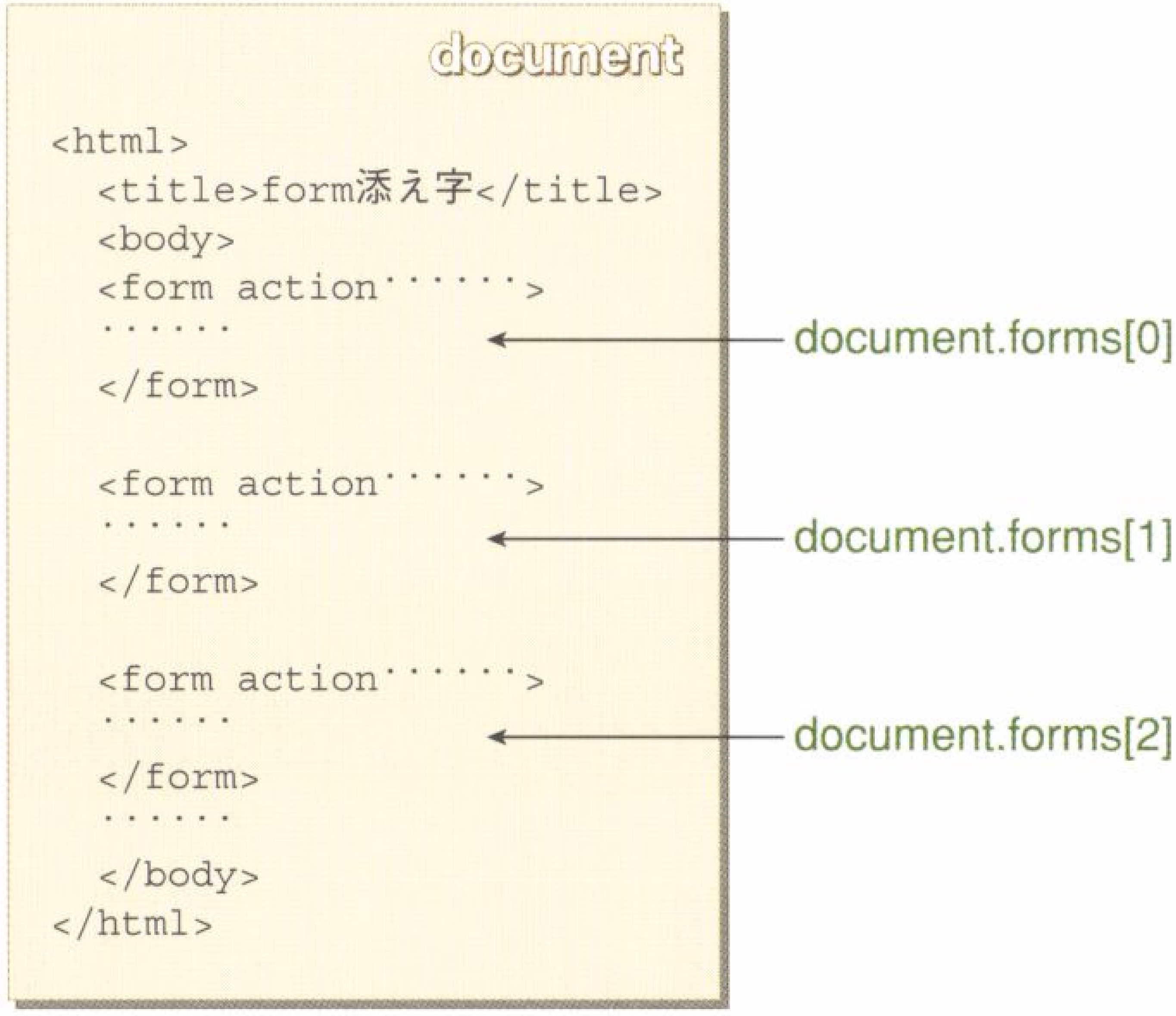
- フォーム全体　→　forms
- フォームの各要素　→　elements

また、フォームとフォームの要素は1つのWebページ内で複数存在する可能性があります。そのため、forms、elementsを参照するときには何らかの形で区別をしなくてはなりません。JavaScriptには、その区別に「添え字」と「名前」を使った2種類の方法があります。

■添え字によるアクセス

1つ目は添え字を使った方法です。フォームはWebページ内で定義された順に番号が振られていきます。またフォームの各要素はフォーム内で定義された順に番号が振られます。この番号を使って区別するのが添え字による方法です。

フォームへのアクセスは次のようになります。

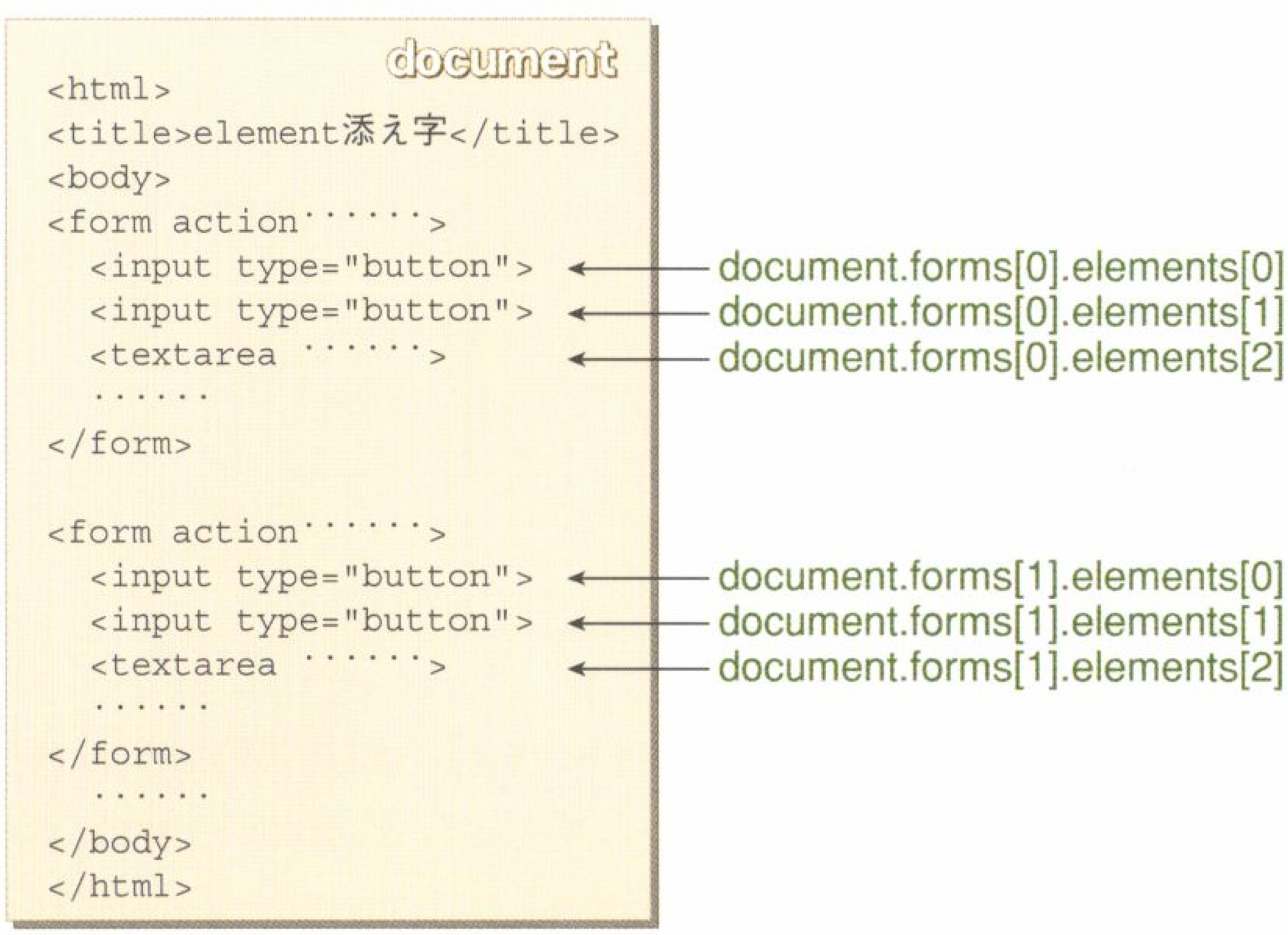


このようにWebページ内で<form>タグが出てきた順に0から番号が振られていきます。これらのフォームの参照は、以下のような記述になります。

Webページの中の(document)の1番目のフォーム(forms[0])

もしもフォームの配置されている場所がフレーム内などの場合は、forms[n]の前の部分にframes[n]が入ります。

フォームの要素(element)へのアクセスは次のようになります。



フォームの各要素へのアクセスも、フォームのときと同様にフォーム内で記述された順に0から番号が振られます。参照は以下のようになります。

Webページの中の(document)の1番目のフォーム(forms[0])の1番目の要素(elements[0])

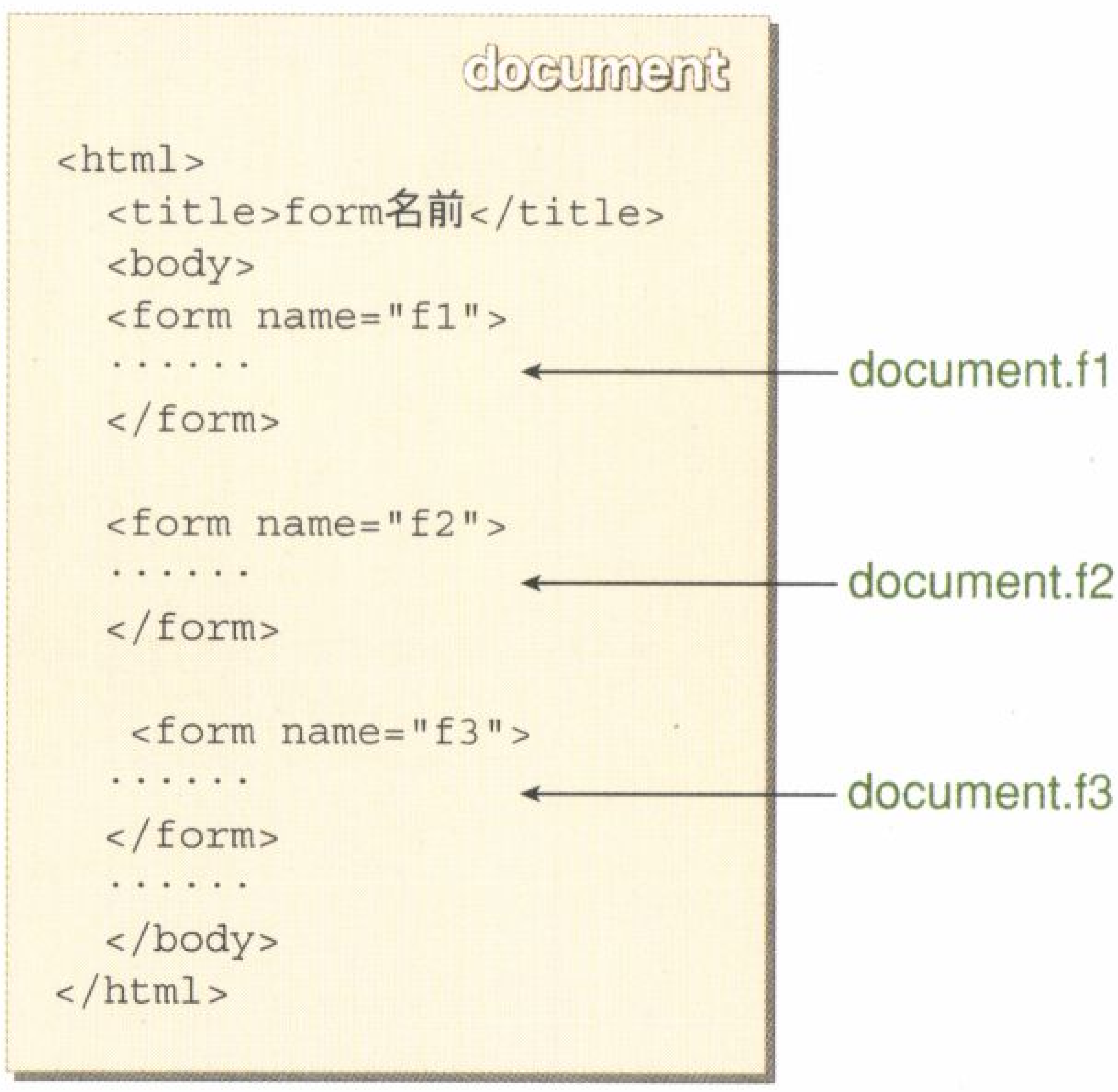
実際の記述では、次のようになります。

```
value1 = document.forms[0].elements[1].value;
str = window.frames[0].forms[2].elements[0].value;
```

■名前による参照

フォーム、フォームの要素の参照には、添え字だけでなく名前による参照もサポートされています。名前による参照の場合には、それぞれの要素のname属性によって付けられた名前を参照します。そのため、同じWebページ内のフォームに同じ名前を使ってはいけません。また同様に、あるフォーム内でフォームの要素に同じ名前を使ってはいけません。

名前による参照は次のようになります。



このように、添え字によるアクセスよりも、名前による参照のほうが理解しやすいと思います。少なくともこのフォームの要素が、Webページ内の何番目なのかといったことは数えなくてもよいからです。

また、スクリプトを書いた後からフォームやフォームの要素を追加しても、番号でなければ参照のための部分は変更しなくても問題はありません。制作途中にname属性によって名前を付けるという作業が必要となりますが、フォームの数が増えれば増えるほど、この作業によるメリットが大きくなることでしょう。個人的には名前による参照をお勧めします。

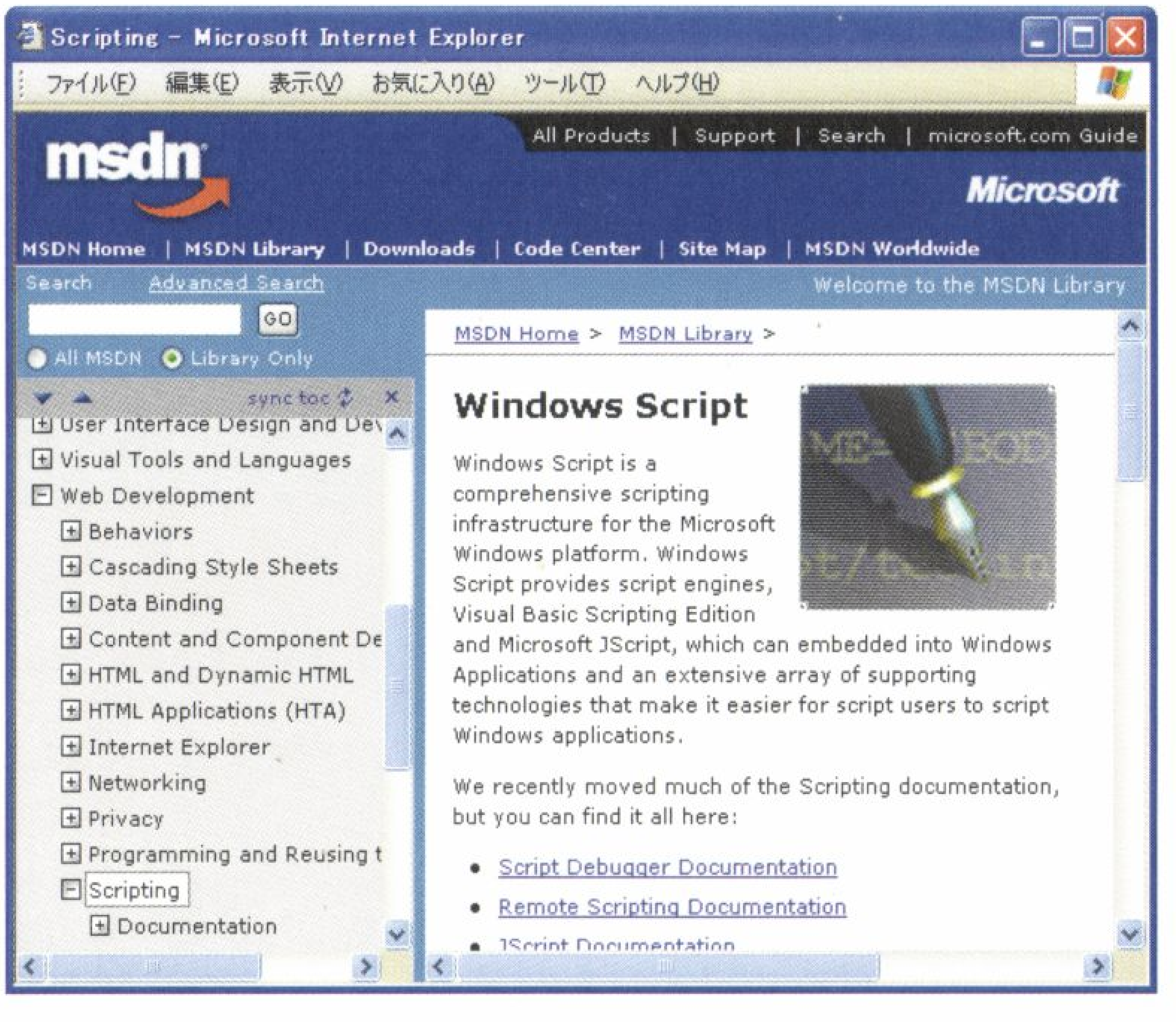
なお、添え字による参照と名前による参照は並存できますので、名前を付けておいても番号で参照することができます。

フレーム

NN2の登場以降急速に広まり、Webページの作成に大きな影響を与えたのがフレームです。ただ、最近ではサーバーサイドでダイナミックにWebページを生成するサイトも多いため、以前に比べるとフレームの使用は減っているようです。

使い方としては、Webページを左右に2分割して左側にメニュー、右側に情報を表示したり、一番下に広告バナー用のフレームを作ったりと、さまざまなバリエーションが試されています。

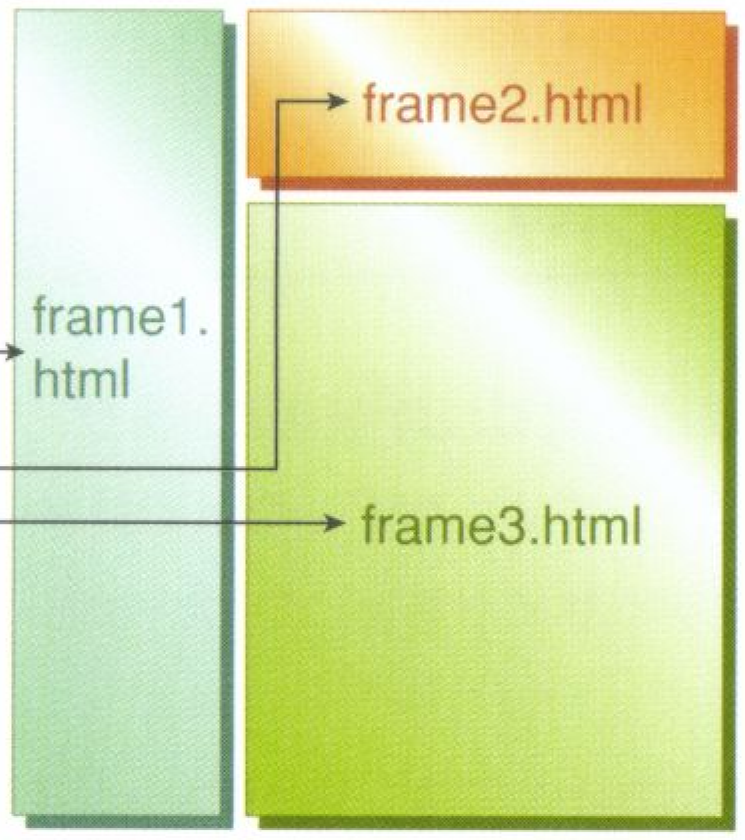
▼フレームを使ったWebページの例



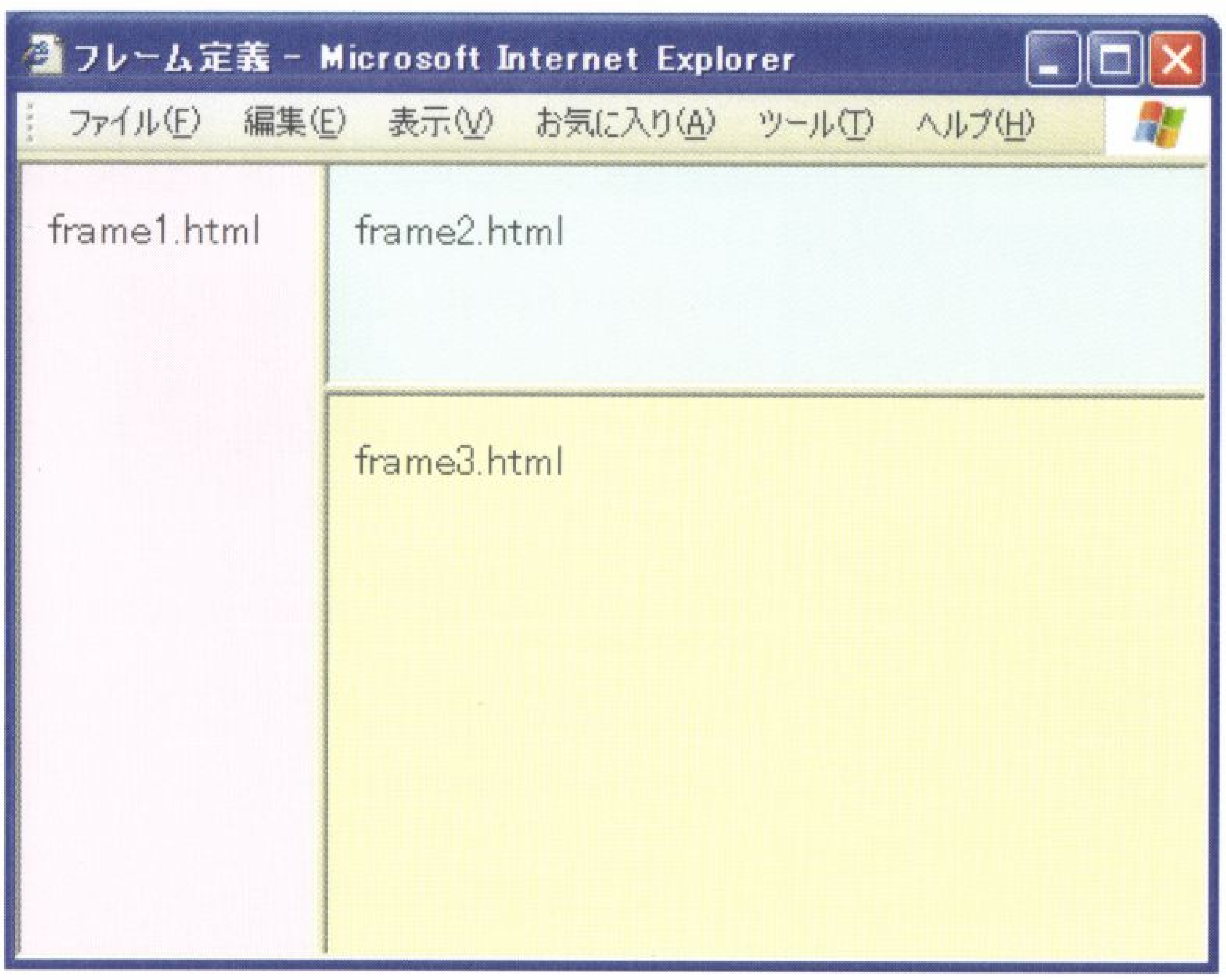
JavaScriptでは、フレームもインタラクティブに制御することができます。ただしフォームを制御するのと同様に、事前知識は必要となります。しかし、フォームをしっかりと理解できているならば、ほとんど同様に扱うことができます。

フレームを作るには、必ずフレームの構造を定義するファイルが必要となります。たとえば次のようになります。

```
<html>
 <title>フレーム定義</title>
 <frameset cols="150,*">
  <frame src="frame1.html">
  <frameset rows="150,*">
   <frame src="frame2.html">
   <frame src="frame3.html">
  </frameset>
 </frameset>
</html>
```

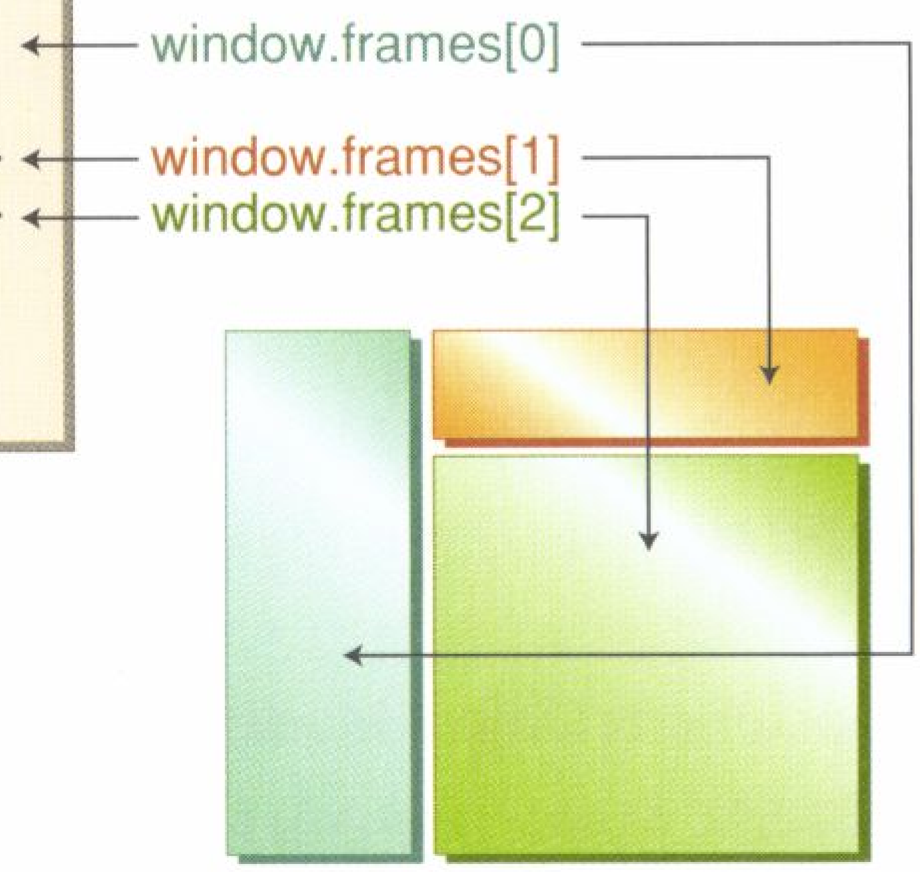


フレームの指定についての詳しい解説は、HTMLの解説書を参照してください。実際にフレームを使って表示させてみると、次のようになります。



ここでフレームをどのように参照するかというと、フォームのときと同じようにHTML文で、<frame>タグが定義された順に0から番号が振られます。この番号を使って、次のように参照します。

```
<html>
 <title>フレーム添字</title>
 <frameset cols="150,*">
  <frame src="frame1.html">
  <frameset rows="150,*">
   <frame src="frame2.html">
   <frame src="frame3.html">
  </frameset>
 </frameset>
 </html>
```



また、フレームもフォームと同様に名前によって参照することができます。フレームの場合は<frame>タグのname属性を使って名前を付けます。ただし、それぞれのフレームには必ず異なった名前を付けてください。同じ名前にするとWebページ中でフレームを一意に特定することができなくなってしまいます。

参照するための書式は以下のようになります。

```
window.フレーム名
```

2-10 DOM

最近では、フォームやフレームを使うだけでなく、Webページをダイナミックに動かしたり、特殊効果を付けたりするために、さまざまな要素を操作する必要があります。このようなときに必要となるのが、DOM(Document Object Model)という考え方です。

DOMは、Webページ内の要素を階層構造で管理しようと考案されたものです。この規格自体はW3Cで制定されています。現在、DOM Level1とDOM Level2が勧告案になっています。なお、DOM Level2は5つの仕様から構成されますが、すべては紹介しきれませんので、ここでは1つだけ紹介します。

●Document Object Model (DOM) Level 1 Specification

URL▶ `http://www.w3.org/TR/REC-DOM-Level-1/`

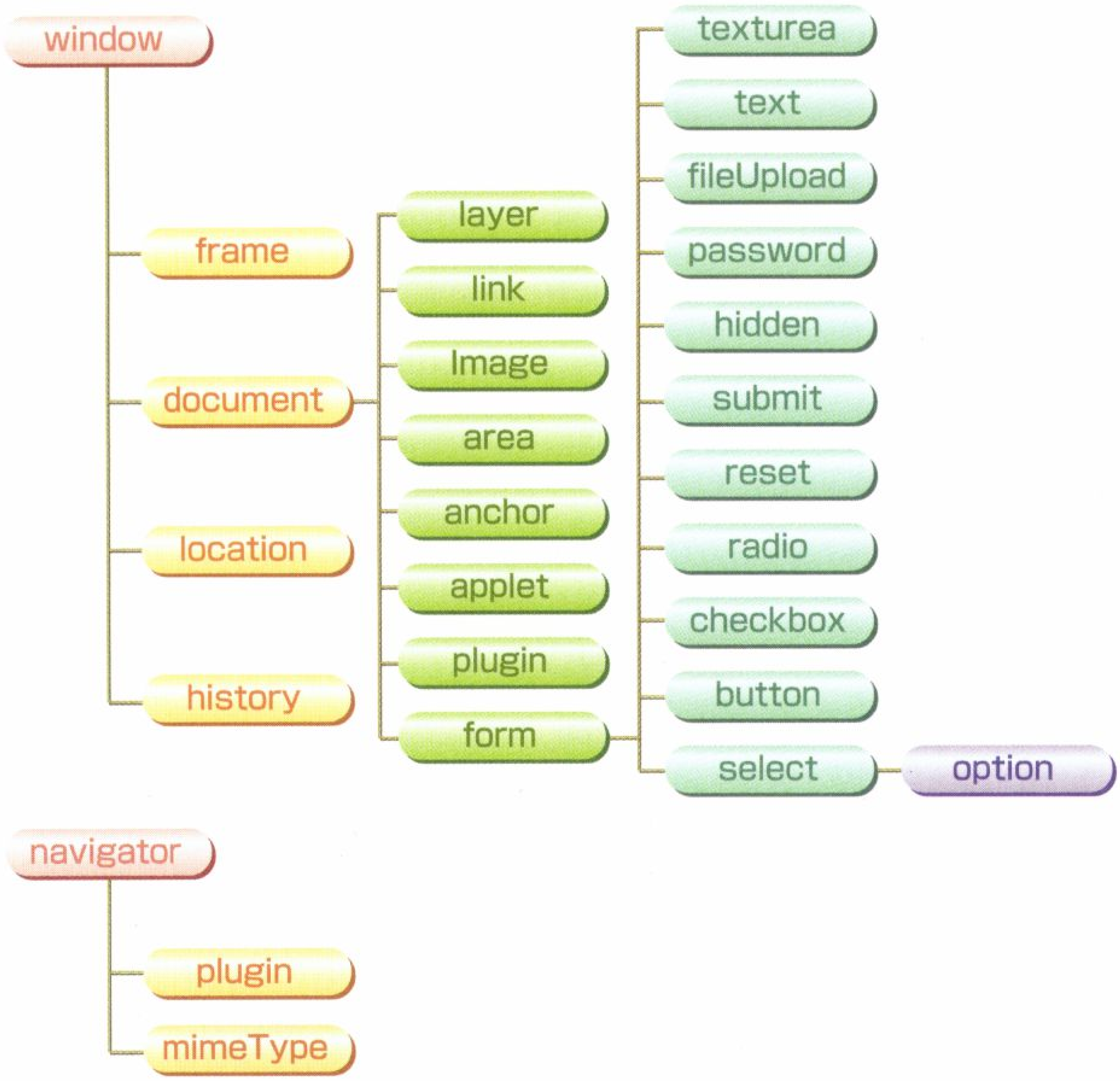
●Document Object Model (DOM) Level 2 Core Specification

URL▶ `http://www.w3.org/TR/DOM-Level-2-Core/`

上記の勧告を見ると、かなりボリュームがあり、非常に面倒に思うはずです。しかし、JavaScriptとHTMLを使ってWebページを作るという観点からすると、使う部分は少なく、実はさほど難しいものではありません。実際に一部ならJavaScriptのコードではよく使っています。しかし、盲目的にDOMを使うよりも、若干内容についても知っておいたほうがスクリプトを理解するのが容易になるので、簡単に解説しておきます。

DOMの構造

DOMの構造は、実は非常に簡単で、windowをトップにした階層構造の下に、Webページで使うオブジェクトが配置されているというものです。



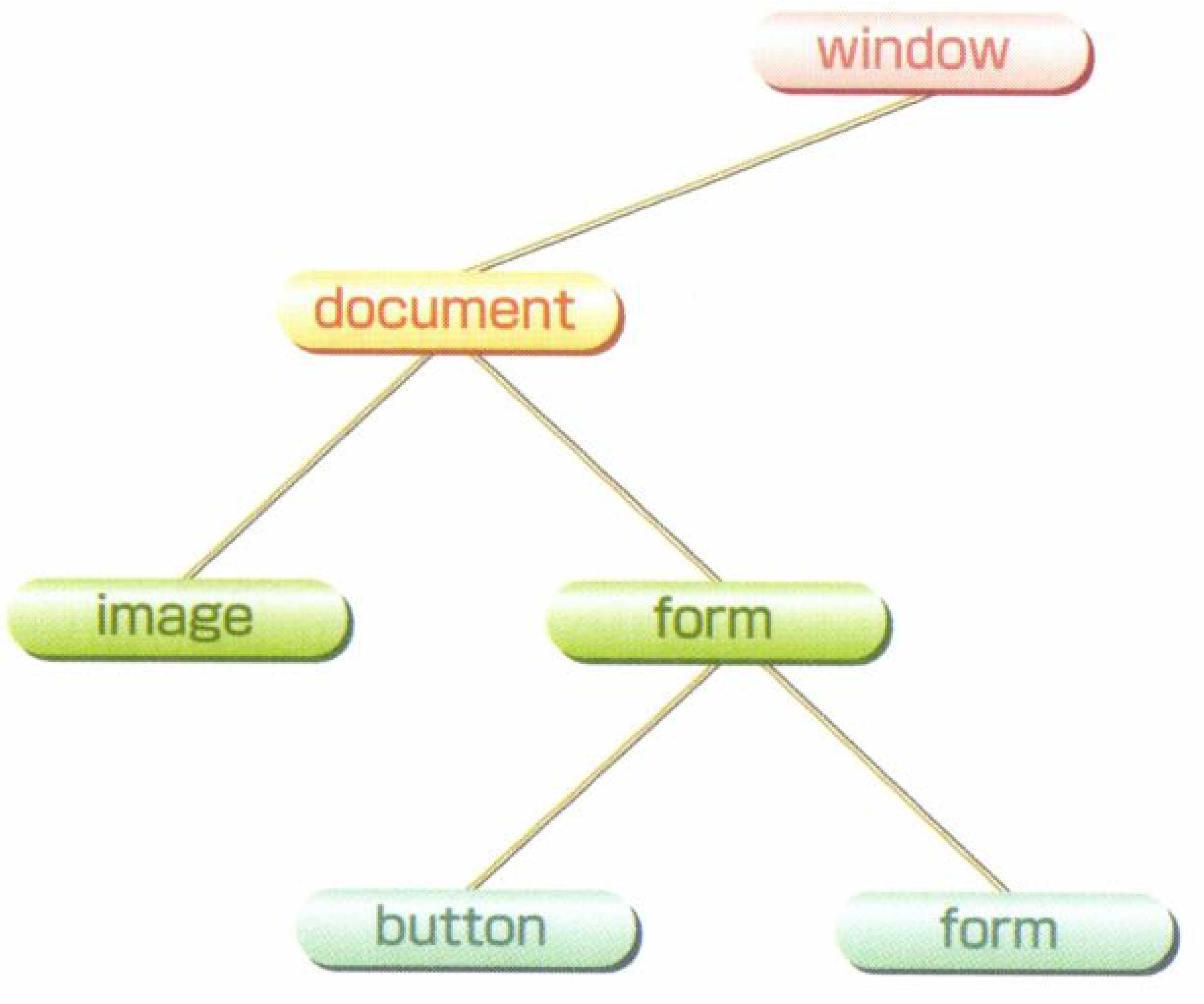
このように階層構造になっているのは、Webページ内に散在している各種オブジェクトに容易にアクセスできるようにするためです。この構造がわからないと、JavaScriptからもオブジェクトを指定することができなくなるため、操作ができないことになります。

たとえば、次のようなHTMLがあったとします。

```
<html>
<title>DOM sample</title>
<body>
<h3>Sample</h3>
<hr />
<img src="image1.gif" />
<form name="myForm">
```

```
    <input type="button" value="click1" />
    <input type="button" value="click2" />
</form>
</body>
</html>
```

このときのDOM構造は、次のようになっています。

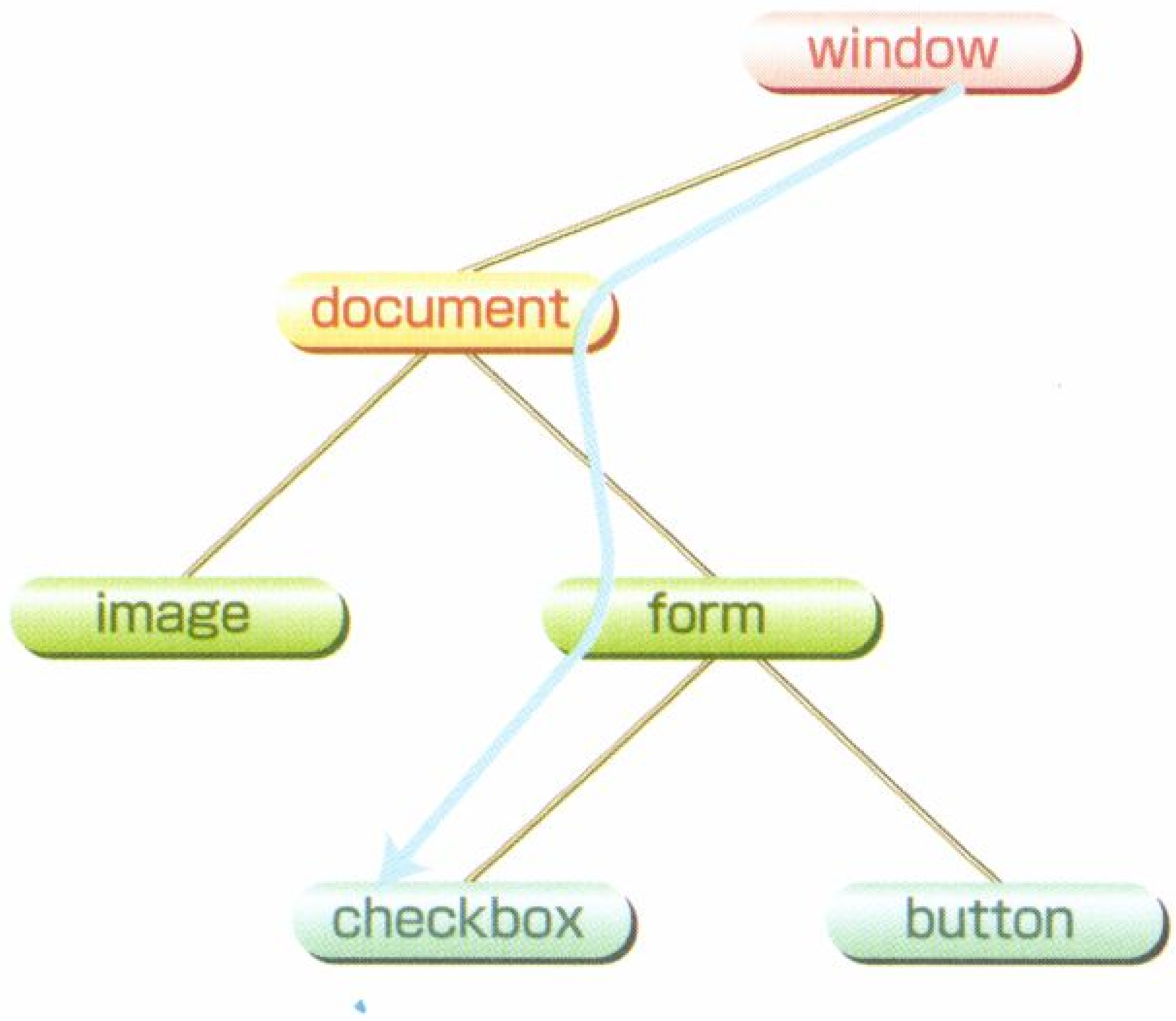


このように、HTMLは必ずDOMにより表現できます。そのため、JavaScriptなどのスクリプト言語でHTML内のオブジェクトを操作しようとしたときには、必ず意識しなくてはならないことになります。

オブジェクトの特定

DOMを使って各種オブジェクトを特定するには、どのようにしたらよいのでしょうか？　これも、実はさほど面倒なものではありません。

たとえば、次のようなDOM構造を持ったHTML文書があり、checkboxフォームのチェックの状態をコントロールしたいとします。



この場合、スクリプト中でのcheckboxフォームの表し方は、以下のようになります。

```
window.document.forms[0].elements[0]
```

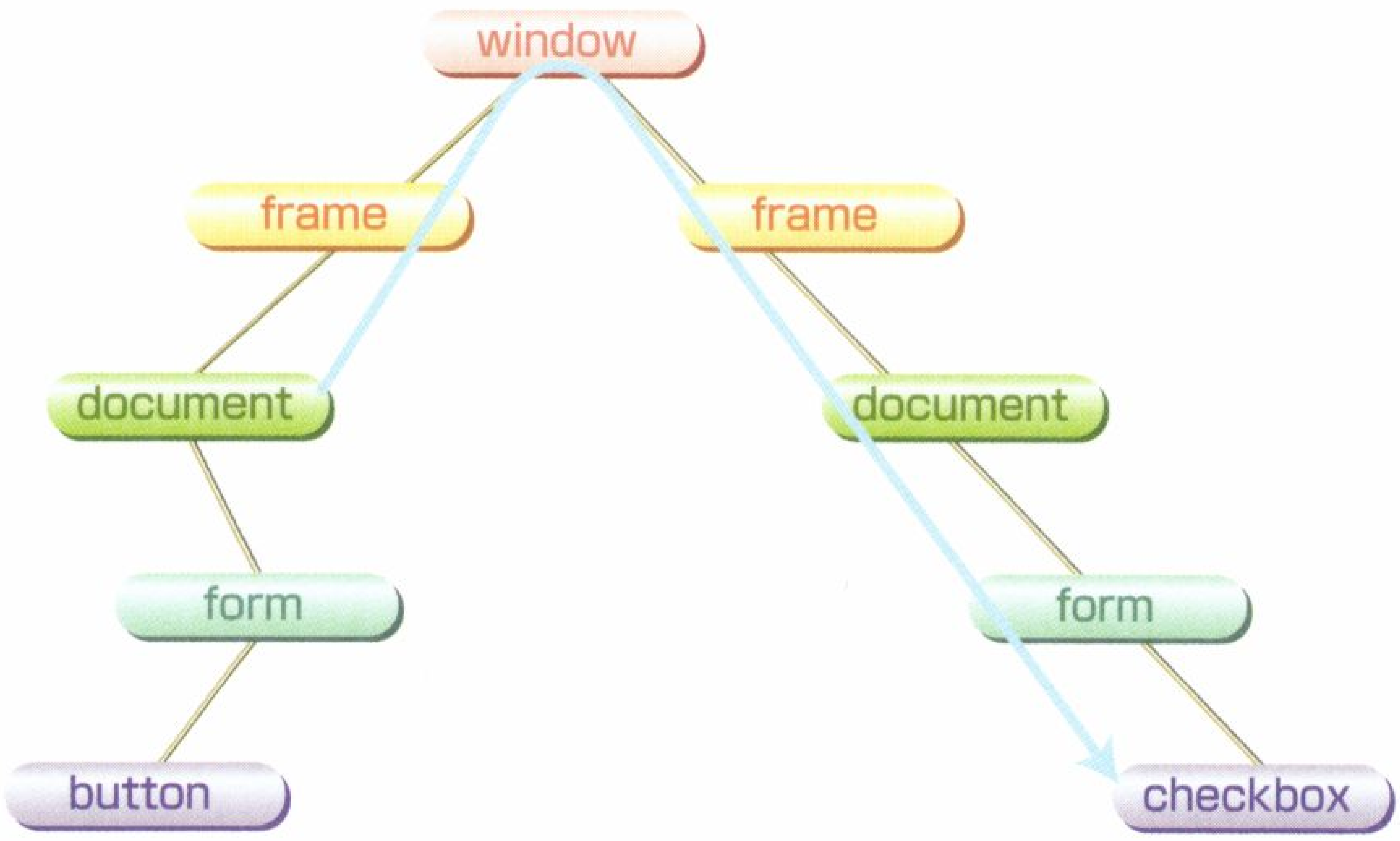
windowオブジェクトは省略してもよいので、次のように記述することもできます。

```
document.forms[0].elements[0]
```

こういった表現なら、JavaScriptをちょっと使っている人ならば理解がしやすいのではないでしょうか？

階層構造のトップから書き連ねていけば、DOMを表現できたことになります。formsとelementsの添え字は、Webページ内で何回目に出てくるフォーム、またはフォーム要素なのかということを表しています。

また、DOMを操作するときにwindowからはじめずに、特定の場所から相対的に記述することもできます。たとえば、次ページのようなDOMを持つHTMLがあり、DOMツリーの左側にあるドキュメント内に記述されているスクリプトから、右側にあるフォーム内のCheckboxフォームを特定したいとします。



このときはまず、以下のような記述が考えられます。

```
window.frames[1].document.forms[0].elements[0]
```

これは先に説明した、トップのwindowオブジェクトから階層構造を下ってきたものです。ここで、これを相対的に表現すると以下のような記述も可能となります。

```
parent.frames[1].document.forms[0].elements[0]
```

parentとは、自分より1階層上のオブジェクトを表しています。

それでは、ここまでの解説をふまえて、実際のスクリプトの例を挙げておきしょう。

dom.html

```
<html>
<title>dom005</title>
<frameset cols="30%,70%">
   <frame src="left.html" />
```

```
    <frame src="right.html" />
</frameset>
</html>
```

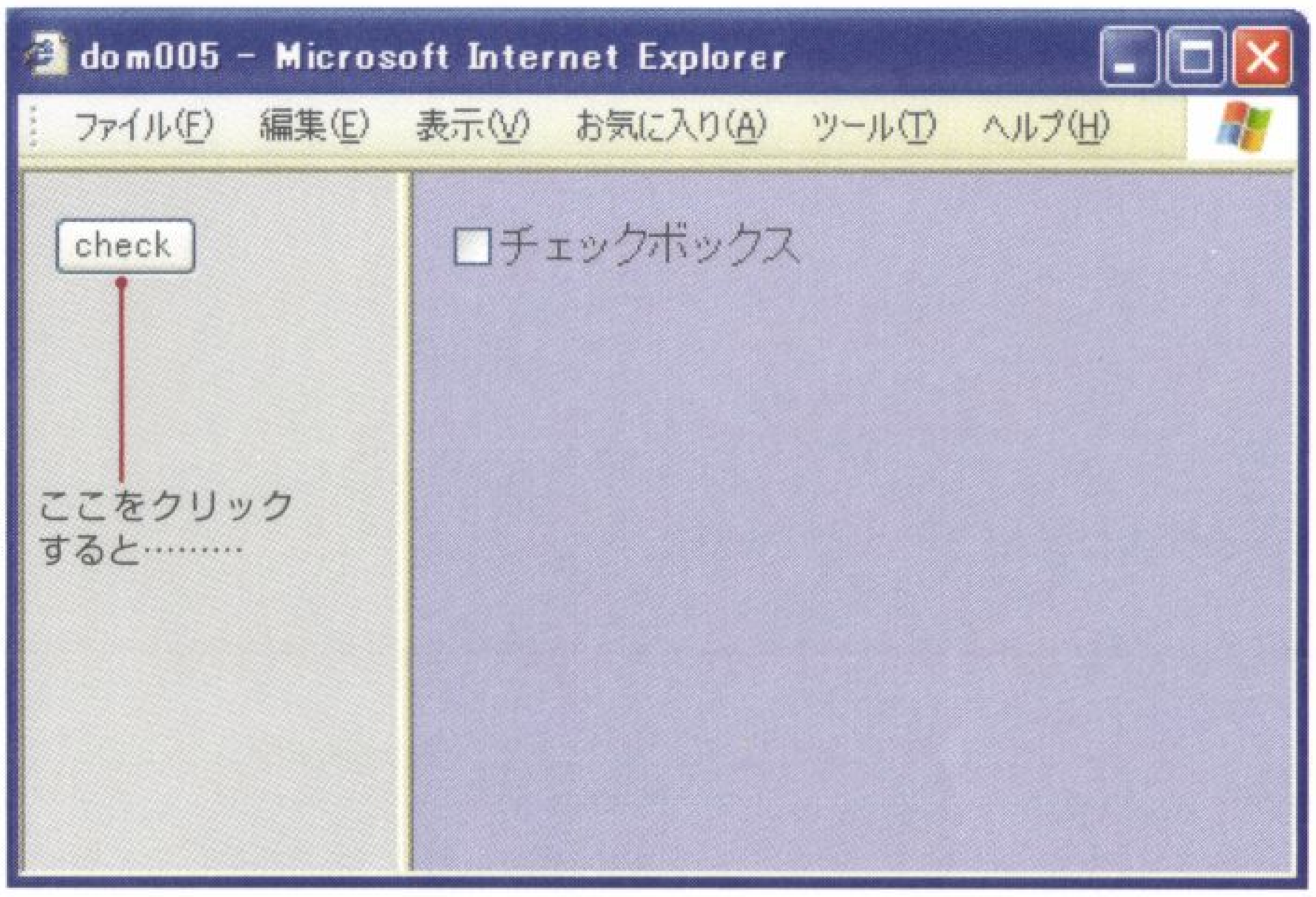
left.html

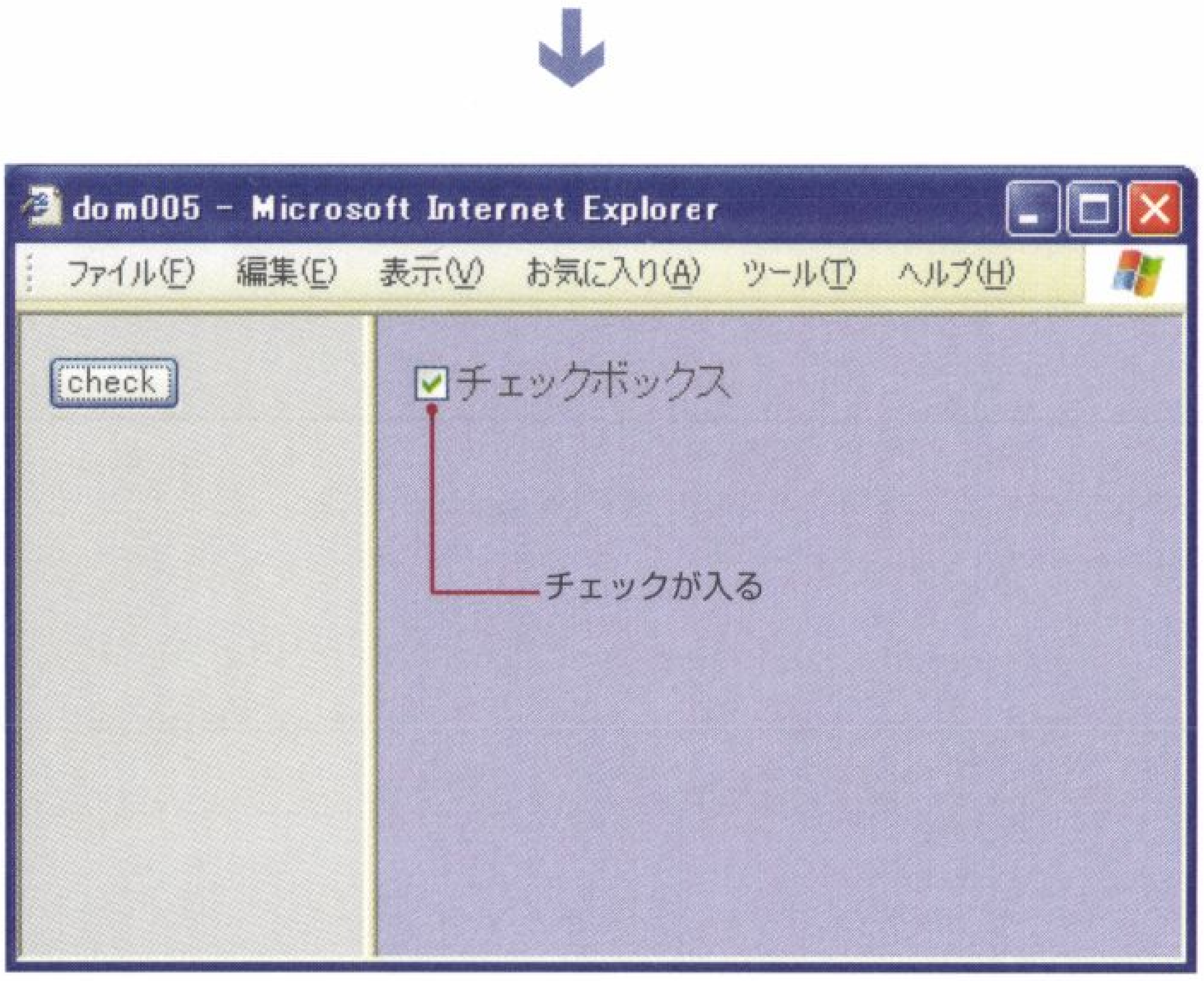
```
<html>
<title>dom left</title>
<script language="javascript">
<!--
function chechAtRight(){
    parent.frames[1].document.forms[0].
⇒      elements[0].checked = 'true';
}
//-->
</script>
<body bgcolor="#dddddd">
<form>
    <input type="button" value="check"
⇒      onclick="chechAtRight()" />
</form>
</body>
</html>
```

right.html

```
<html>
<title>dom right</title>
<body bgcolor="#c0c0dd">
<form>
    <input type="checkbox" />チェックボックス
</form>
</body>
</html>
```

この例のスクリプトを実行すると、次のようになります。





Webブラウザの画面の要素とDOMの考え方について解説してきました。しかし、DOMの仕様はこれだけではありません。DOMでは、オブジェクトの要素を操作する関数や、オブジェクトについて調べたりする関数もあります。しかし、本書では誌面の都合もあり紹介はしません。

より詳細な情報は、DOMの仕様書や、Microsoft社、Netscape社が提供している資料などをご覧ください。

2-11

イベント

イベントとはWebページを見ているユーザーが行った操作のことを指します。イベントから特定のものに対してアクションを起こすことにより、さまざまな処理を行うようにスクリプトを記述します。

イベントにはどのようなものがあるかを具体的に挙げると、次のようなものがあります。また、利用可能なイベントを表で一覧にしておきます。

- フォームのボタンをクリックしたとき
- ハイパーリンクになっている文字列をクリックしたとき
- Webページがロードされたとき

●JavaScriptで利用可能なイベント

イベント	いつ発生するか	イベントハンドラ
abort	ユーザーがイメージの読み込みを中止した場合に生じる。たとえば、イメージのダウンロード中に、他のリンクやWebブラウザの中止ボタンをクリックしたときなど	onAbort
blur	Webブラウザ本体、Webページ内のフレーム、フォームのボタンなどから、ユーザーのフォーカスが離れた場合に生じる	onBlur
click	ユーザーがリンク上やフォーム要素上でクリックした場合に生じる	onClick
change	ユーザーによって要素(フォームフィールド)の値が変更されたときに生じる	onChange
error	ドキュメント、イメージの読み込みエラーが起きた場合に生じる	onError
focus	Webブラウザ本体、Webページ内のフレーム、フォームのボタンなどにユーザーのフォーカスが与えられた場合に生じる	onFocus
load	WebページがWebブラウザにロードされたときに生じる	onLoad
mouseout	ユーザーが領域の外にカーソルを移動させたときに生じる	onMouseOut
mouseover	ユーザーが領域にカーソルを移動させたときに生じる	onMouseOver
reset	ユーザーがresetボタンをクリックしたときに生じる	onReset
submit	ユーザーがsubmitボタンをクリックしたときに生じる	onSubmit
unload	ユーザーがWebページから出るときに生じる	onUnload

イベントハンドラ

Webページに対して、ユーザーが操作することで、さまざまなイベントが発生します。ユーザーがWebページ上の何もないところでクリックしたとしてもイベントは発生しています。つまり、ユーザーのほとんどの操作に対してイベントが発生しているのです。

しかし、通常は何も起こりません。これはそれぞれのイベントに対応した処理が存在しないからです。イベントの中から目的の場所で目的のイベントを取り出すための仕組みを、JavaScriptでは「イベントハンドラ」と呼んでいます。JavaScriptでのイベントハンドラは、HTMLタグを用いて記述する方法と、オブジェクトを使って記述する方法があります。

主なイベントハンドラは、前ページの表に示したとおりです。

■HTMLタグを用いたイベントハンドラの記述

HTMLタグを用いたイベントハンドラの書式は、以下のようになります。

```
<HTMLタグ イベントハンドラ = "JavaScriptプログラム">
```

上記のような記述で、ユーザーがHTMLタグの領域上でイベントハンドラに対応したイベントを起こしたときに、指定したJavaScriptのプログラムが実行されます。

イベントハンドラに与えるJavaScriptプログラムは、さまざまな書き方をすることができます。直接プログラムを記述することもできますし、関数を指定することもできます。関数を指定するには、先に関数を定義しておく必要があります。

たとえば事前にprintstatus関数が定義してある場合、それを呼び出すには次のような記述となります。

```
<input type="button" onclick="printstatus()" />
```

この記述では、フォームのボタンをクリックしたとき(onclick)は、printstatus関数が呼び出され、関数内に記述されている処理が行われます。

JavaScriptのコマンドを直接記述する場合は、次のような記述になります。

```
<input type="button" onclick="
   if(value>=4){alert('You are Right!');}
   else{alert("Oh no!");}
" />
```

また複数のコマンドを使用したい場合には、「;」(セミコロン)で分離することで実装できます。

```
<input type="button" onclick="
   appName = navigator.appName;
   alert('あなたのWebブラウザは'+appName+'ですね。');
" />
```

このように、イベントハンドラに対してのJavaScriptプログラムの定義には、2種類の方法があります。では、どちらを使うのがよいのでしょうか？　これはイベントハンドラに対応した処理がどのようなものかで決まります。

もしもHTMLファイル内で1回しか行われない処理で、処理のスクリプトが短いものならば、HTMLタグ内のイベントハンドラに直接処理を記述してしまうのがよいと思います。しかし、1回しか行われない処理でもあまりにスクリプトが長い場合には、視認性やメンテナンス性のことを考えて、関数を定義してイベントハンドラで呼び出すのがよいでしょう。

たとえば、次のようなコードは視認性がよくありません。

```
<input type="button" onclick="
   if(value>=4){
   document.write(navigator.appName);
   document.write(navigator.appversion);
   }
   else if(value==3){
   alert('このページがちゃんと動く？');
   document.write('<A href="****">');
   }
   else{

   ……
```

```
        }
" />
```

この視認性の悪さを改善するには、スクリプトの部分を関数にして、呼び出す形式にします。

```
<script language="javascript">
function check(){
   if(value>=4)
      document.write(navigator.appName);
      document.write(navigator.appversion);
   }
   else if(value==3){
      alert('このページがちゃんと動く？');
      document.write('<A href="****">');
   }
   else{

   ……

   }
</script>
<form>
   <input type="button" onclick="check()" />
</form>
```

また、呼び出す処理を同じWebページ内で何度も行うならば、関数として定義して呼び出す形式にすると、記述量が少なくなるため視認性やメンテナンス性が向上します。

たとえば、Webページ内に複数のフォームを用意し、ユーザーに数値だけを入力させたいとします。そのとき、入力された内容が本当に数値かどうかをチェックする関数をフォームごとに書いていたら、非常に面倒なことになります。そこで、数値かどうかをチェックする関数を1つ記述しておき、フォームからそれを呼び出すことで記述量を減らせます。

なお、イベントハンドラはどのタグにも属性として付けられるわけではありません。JavaScriptのリファレンスを確認してください。

■オブジェクトを使ったイベントハンドラの記述

イベントハンドラの記述として、オブジェクトを使った記述もあります。HTML要素(タグ)にイベントハンドラを設定する方法は、通常イベントハンドラの対象を特定できる場合に用いられます。その代表的なものとして、各種フォームがあります。

これに対して、Webブラウザ全体のどこで起きてもよいようなイベント(たとえばonkeydownやonloadなど)は、特定のHTML要素に対してイベントハンドラを設定するのが難しいことがあります。そんなときに使うのが、オブジェクトに対してイベントハンドラを設定する方法です。

書式は、以下のようになります。

```
オブジェクト.イベントハンドラ = 命令;
```

実際の記述例としては、次のようなものになります。

●ウィンドウがリサイズされたときresize_win関数を呼び出す

```
window.onresize = resize_win;
```

●ドキュメント上でキーが押されたらkey_down関数を呼び出す

```
window.document.onkeydown = key_down;
```

このようにオブジェクトを指定し、それにイベントハンドラを設定することができます。なお、通常の関数の呼び出しは引数を取らない場合でも「( )」が付くのですが、この書式を用いたときには「( )」を付けませんので注意してください。

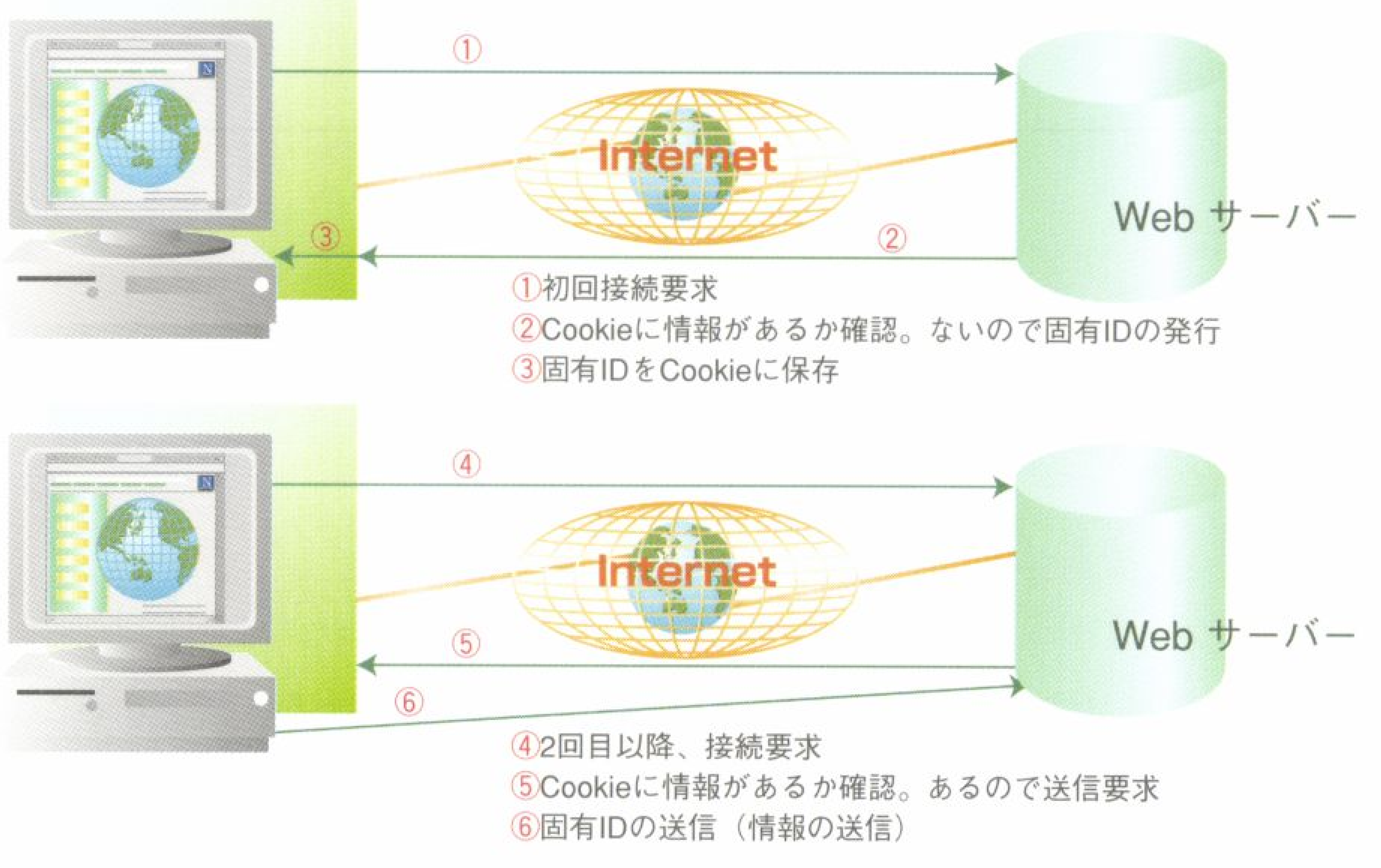
2-12 Cookie（クッキー）

Cookieとは

Webブラウザは、通常ユーザーが使っているマシンのローカルファイル(クライアント側)に情報を保存することができません。これはWebページからダウンロードしてきたプログラムなどを勝手に保存できるようになっていると、セキュリティ上問題があるからです。しかし、JavaScriptを用いると「Cookieファイル」と呼ばれるものをクライアント側に保存することができます。

Cookieは非常に単純なもので、ある決められたフォーマットで、ある決められた場所にだけ、テキストで情報を保存できるというものです。そのため、Webサーバーに接続してきたユーザーに発行したIDや、Webページを使う上での設定情報を保存するときなどに使います。

▼Cookieの動作



従来、Webサーバーは、ユーザーが情報を入力してこない限り、接続してきたクライアントが誰なのかを特定することができなかったため、特定のユーザーに対して、特定の動作を行ったりすることは実現困難でした。しかし、それではイントラネット上などでの業務用Webアプリケーションを作成することができません。このような問題とセキュリティの両方をクリアしたのがCookieです。

Cookieの動作は前ページの図のようになります。JavaScriptのプログラムは、あるユーザーがWebサーバーに接続したとき、そのWebサーバーが発行したユーザーを特定するIDが保存されているかどうかを確認します。もしもIDがなかった場合、Webサーバーは次回以降そのユーザーを特定できるように固有のIDを発行しますので、これをCookieファイルとして保存しておきます。これで2回目以降の接続では、固有のIDの確認時に、IDをWebサーバーに送ることができます。この固有IDを使って、ユーザーそれぞれにカスタマイズしたWebページを送ったり、以前に設定された値で、ユーザー固有の処理を行ったりすることができます。

なお、実際にはWebブラウザごとにCookieは保存されるので、ユーザーではなく、あるPCのWebブラウザを特定していることになります。

Cookieの制限

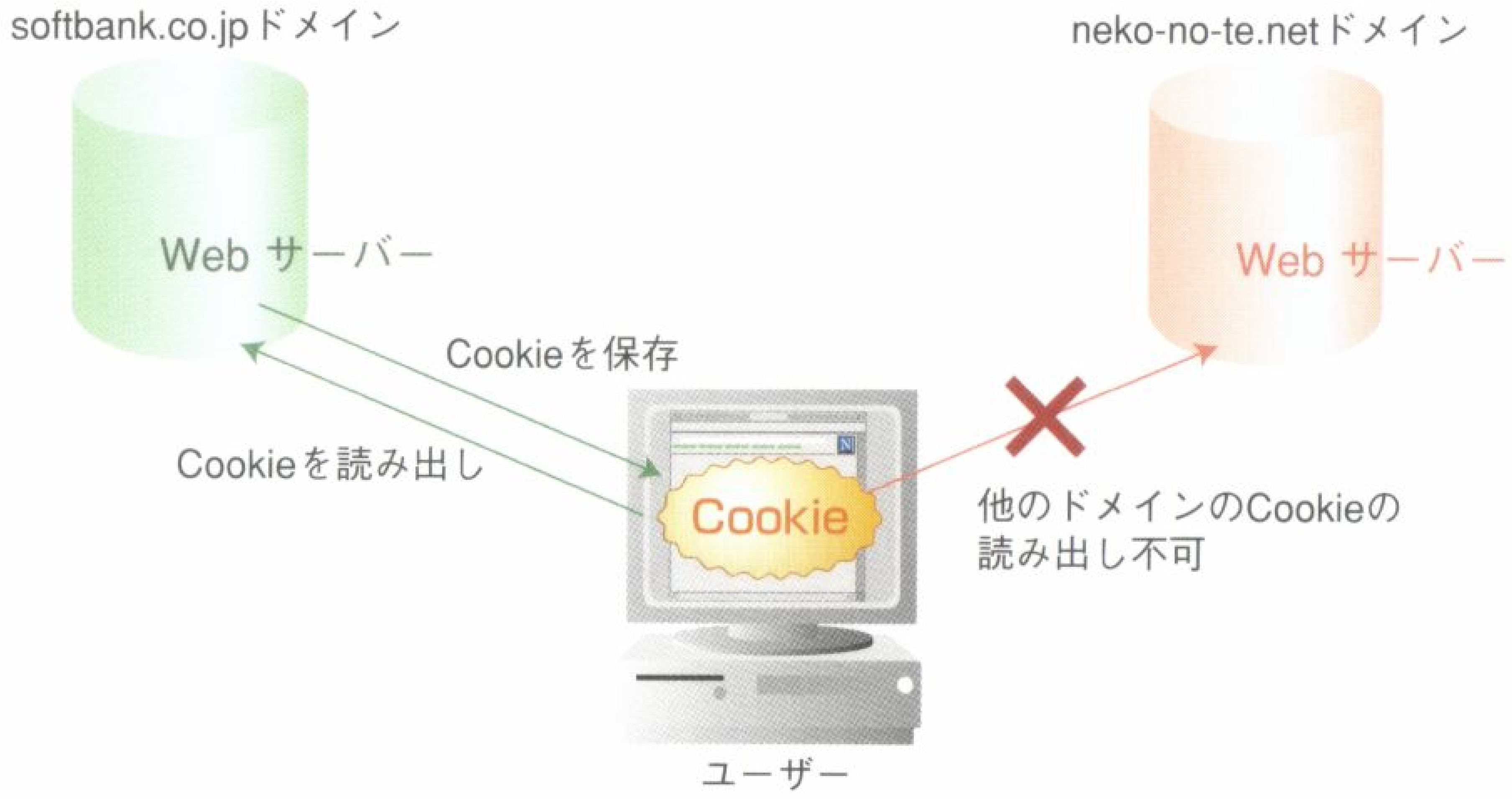
Cookieは非常に制限された形でクライアント側に情報を保存することができます。その制限とは、以下のようなものです。

- 保存先が限定されている
- 一定の書式でしか書き込みができない
- 保存形式はテキストのみである
- ユーザーに関しての情報は送信されない
- 他のドメインのCookieを読み出せない

まず保存先は、ある特定のフォルダに固定されています。そこ以外には保存することができません。保存できる形式はテキストのみであり、その記述方法も決まっています。また、重要なことなのですが、接続した人の電子メールアドレスやハードディスクの内容といった具体的な情報は一切集めることができません。これは、個人の情報を勝手に読み取れないようにするためと、ウイルスなどローカルマシンに対して攻撃を加えるものは保存できないようにするためです。こういった点で、プライバシーの問題も解決されているといえます。

また、他のドメインが保存したクッキーを読み出すことはできません。ここで、ドメインとは、URLのドメイン部分に相当します。たとえば、あるWebページのドメイン名で登録されたクッキーの内容は、そのドメインからしか読み出すことがないのです。

▼クッキーは保存したドメインしか読み出せない



Cookieファイルの保存場所

Cookieファイルはどこに保存されているのでしょうか？　これは使っているOSやWebブラウザによって異なります。たとえば、IEとNNの場合には、それぞれ次のフォルダに保存されています。

●IEとNNのOS別Cookieファイルの保存場所

ブラウザ	OS	保存場所
IE6	Windows 98/Me（単独利用）	C:¥Windows¥Cookie¥
	Windows 2000/XP	C:¥Documents and Settings ¥<username>¥Cookie¥
IE5	Macintosh	[初期設定フォルダ]内の[Internet Preference]
NN6	Windows 98/Me	C:¥Program files¥Netscape ¥Netscape6¥User¥<username. ¥cookies.txt
	Windows　2000/XP	C:¥Documents and Settings ¥<username>¥Application data ¥Mozilla¥Profiles¥<username> ¥<セットアップ日時.sit>¥cookies.txt
	Macintosh	[初期設定フォルダ]内の[Netscape Users]内の[ユーザー名]内の[MagicCookie]

この例のように、それぞれのOSやWebブラウザごとに、保存される場所やファイル名は異なりますが、そのことをJavaScriptのプログラムで意識する必要はありません。Cookieはアクセスしたサーバーの名前からファイルを作成、またはキーを作成して情報を保存します。そのため、保存するファイルをいちいち指定しなくてよいのです。

Cookieへの書き込み

Cookieへの書き込みは、documentオブジェクトのcookieプロパティに値を代入することにより行います。

Cookieへの書き込みは、次のような形式で行われます。

```
document.cookie = "キー名 = 値"; expires = 期限(GMT);
path = URLパス; domain = サイトドメイン; secure";
```

この中で"キー名 = 値"だけは必ず指定しなくてはなりません。他のexpires、path、domain、secureは指定しなくても、Webブラウザが適切に設定してくれます。ただし、expiresを設定しないと、Webブラウザの終了とともにクッキーの有効期限が切れます。これは、セッションの終了までが有効期限ということです。

それぞれの属性の意味は、次の表のようになります。

名前	説明
name = 値	Cookieファイルに保存されるデータの変数名と、その中に保持される値。これは省略することができない
expires = 期限	Cookieの有効期限の日時を指定する。この日時以降はCookieは保存されず、サーバーにも送信されない。この日時の指定はグリニッジ標準時で行う。書式は原則としてWdy、DD-MM-YYYY、HH:MM:SSとなる
path = パス	有効なCookieのURLのうち、パスの部分を指定する。指定がなければ、パスはCookieを設定したドキュメントと同じになる
domain = サイトドメイン	有効なCookieのURLのうち、ドメインの部分を指定する。指定しなければ、ドメインはCookieを設定したドメインと同じになる
secure	この属性を記述すると、Cookieを安全な接続だけを使って送信するようになる

記述の例としては、次のようなものがあります。

```
document.cookie = "name=" + myName;
```

```
document.cookie = "name=" + myName + "; expires=Fri,
⇒ 01-Jan-2001 12:00:00; path = '/'; domain =
⇒ 'www.softbank.co.jp'; secure"
```

上の1つ目の属性は、省略したもっとも短い指定、下の2つ目はすべての属性を記述した場合です。

Cookieの読み出し

Cookieファイルへの保存、参照はdocumentオブジェクトの下のcookieプロパティを用います。Cookieは、ドメインごとに保存されます。

```
document.cookie
```

参照は任意の変数をmyDataとすると、次のような記述でCookieに格納されている値を参照することができます。

```
myData = document.cookie;
```

このとき読み出されるのは、Webサーバーと同じ名前で保存されているデータの内容です。読み出されるデータは、キーと値が「=」により連結された文字列です。

```
キー=値;
```

もしも1つのドメインで複数のCookieを保存している場合には、キーと値の連結されたものが、「;」(セミコロン)により複数連結されたものが返されます。

```
キー1=値1;キー2=値2; ……… ;キーn=値n;
```

Cookieの値を実際に使うには、この記述で返されるキーと値の組み合わせを、何らかの形で変換しなくてはなりません。

通常はCookieを読み出すときにCookieの値が空でないか確認してから読み出し、さらに変数名と値を分離する作業が必要になります。

実際にCookieから値を読み出す方法については、SECTION12のサンプルスクリプトを参照してください。

Cookieを削除する

Cookieは有効期限になると自動的に削除されます。しかし、有効期限より前にCookieを削除したくなる場合もあります。このようなときには明示的にCookieを削除しなくてはなりません。

Cookieの削除には特別なメソッドがあるわけではありません。どのように削除するかというと、有効期限を現在の時刻よりも過去にするという作業を行います。まずCookieにセットする過去の有効期限を生成します。有効期限の作成ができたら、それを削除したいCookieの書き込みに使います。

```
document.cookie="name=<削除したいcookie名>"; "expires=
現在より過去の時間";
```

これで削除したいCookieは、有効期限が切れたことになるので削除されることになります。

具体的なサンプルスクリプトは、P.375を参照してください。

2-13 クロスブラウザ実現のために

ここでは、いくつかのWebブラウザで同じように見せたり、処理を分割したりするために必要な基礎知識について解説します。

現在シェアの高いIEとNNの2つのWebブラウザの間には、同じスクリプトなのに表示結果が異なるという互換性の問題があります。しかし、不特定多数の人々を相手にWebサイトを構築しなくてはならない場合、何としてでも複数のWebブラウザの複数のバージョン、かつ、複数のプラットフォームで動作するようにしなくてはならないといったことが起こります。このようなときには、どうやって互換性を保てばよいのでしょうか？

互換性を保つための方針としては、次のようなものが考えられますが、それぞれ、メリット・デメリットがあります。

❶互換性の取れている部分だけでWebページを制作する

メリット	：	制作は比較的容易である
デメリット	：	あまり面白みのないWebページになってしまう

❷Webブラウザごとに完全にWebページを作り分ける

メリット	：	最新の技術を使うことができ、豊富な表現が可能になる
デメリット	：	Webブラウザごとに作り分けることにより、制作コストが高くなる
	：	コンテンツを更新するときに、複数のWebページにまたがって更新しなくてはならない

❸Webページ中で互換性のない部分だけ処理を作り分ける

メリット	：	比較的低いコストで最新の技術を使い、魅力あるページを作成できる
	：	Webブラウザごとに複数のコンテンツを管理しなくても済む
デメリット	：	Webブラウザの特性などがわかった上でWebページを制作し、なおかつ最新技術も知らなくてはならないので、高度な技術が必要となる

このようにさまざまなメリット、デメリットがありますが、どの方法がベストかということではなく、Webサイト構築にあたってどのような要求が発生しているかによって使い分けるべきでしょう。また、最初は❶の共通部分だけを使って制作していくという方法を取っておき、だんだんとWebサイトを成長させ、❷または❸に成長させていくという方法もあります。どの方法を取るかには、他の技術との兼ね合いもあるので、慎重に考慮して手法を選んでください。

Webブラウザの判定

それぞれのWebブラウザで共通に動作する部分だけを使ってスクリプトを制作するのでなければ、少なくともWebブラウザの判定が必要になります。このとき何を基準にWebブラウザを判定すればよいのでしょうか？　JavaScriptで利用できるものを考えたときには、以下のようなものがあるでしょう。

- navigatorオブジェクトからWebブラウザ情報を取得する
- <script >要素で分類する
- DOMの違いに注目する

■navigatorオブジェクトからWebブラウザ情報を取得する

最初に、userAgentプロパティやappVersionプロパティを使ってWebブラウザを判別する方法について解説しましょう。

Webブラウザは、ダウンロードしてきたHTMLファイルに対して、Webブラウザの情報を公開します。そのため、HTMLファイルに含まれているスクリプトは、Webブラウザ情報を取得することができます。このとき取得するのは、navigatorオブジェクトのuserAgentプロパティ、appVersionプロパティなどです。navigatorオブジェクトにより取得できる情報には、以下のようなものがあります。

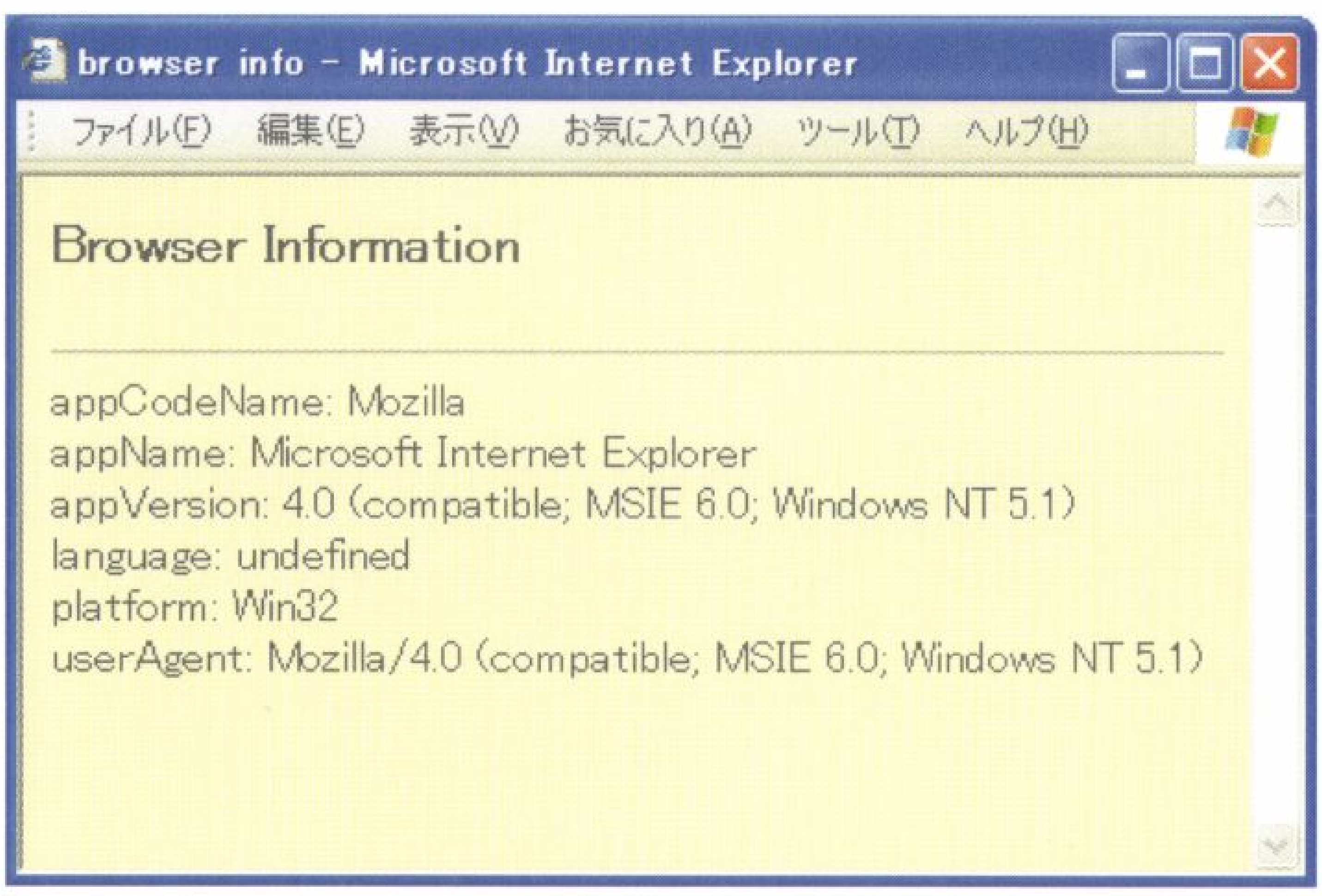
プロパティ	取得できる情報	IEバージョン	NNバージョン
navigator.appCodeName	アプリケーションコード名	3以上	2以上
navigator.appName	アプリケーション名	3以上	2以上
navigator.appVersion	バージョン	3以上	2以上
navigator.language	言語	-	4以上
navigator.platform	プラットフォーム(OS)	4以上	4以上
navigator.userAgent	Webブラウザ情報	3以上	2以上

これらの情報を読み出すには、以下のように行います。

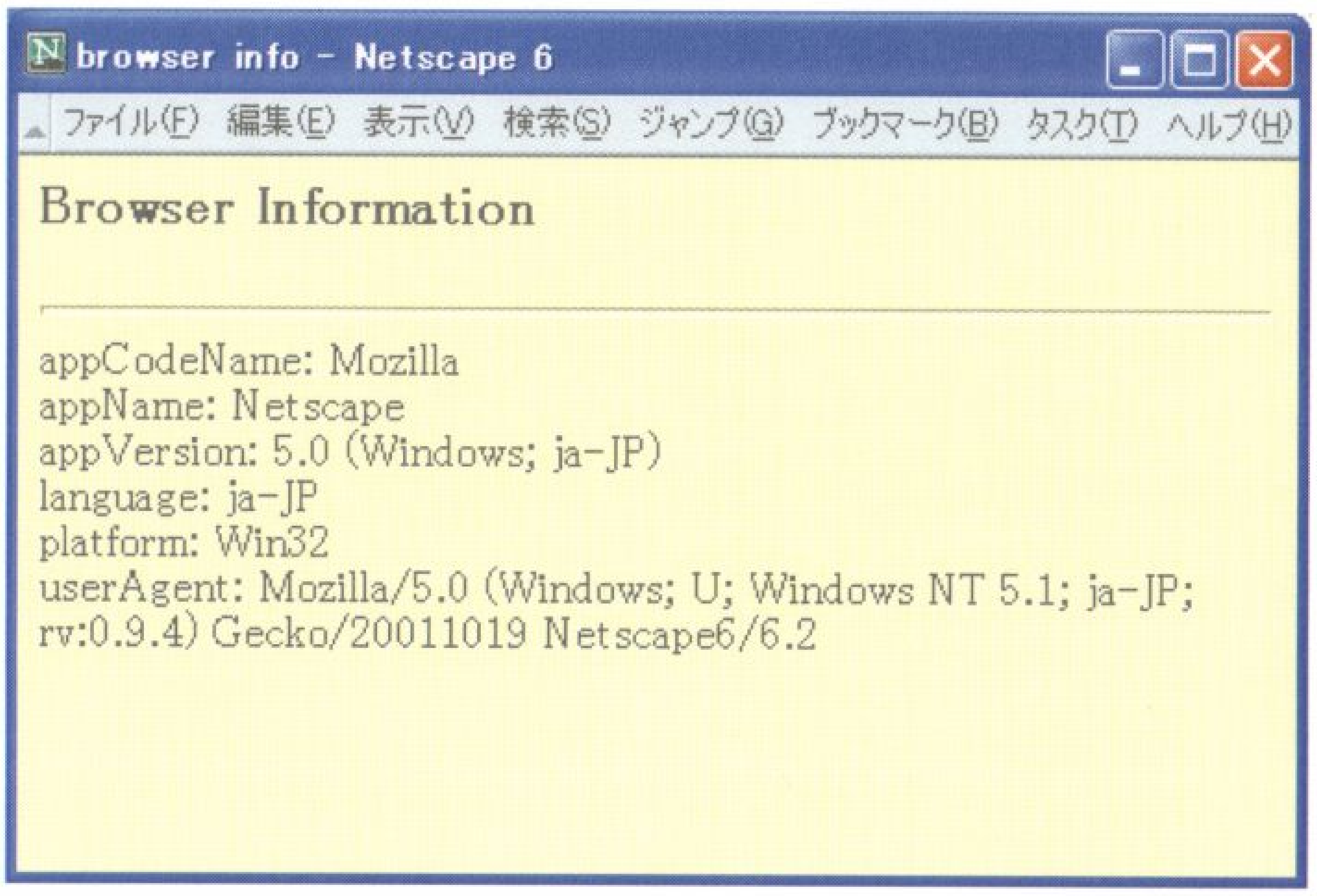
```
<html>
<title>browser info</title>
<body bgcolor="#fff8dc">
<h3>Browser Information</h3>
<hr />
<script language="javascript">
<!--
    document.write("appCodeName: " + navigator.appCodeName + "<br />");
    document.write("appName: " + navigator.appName +"<br />");
    document.write("appVersion: " + navigator.appVersion + "<br />");
    document.write("language: " + navigator.language + "<br />");
    document.write("platform: " + navigator.platform + "<br />");
    document.write("userAgent: " + navigator.userAgent + "<br />");
//-->
</script>
</body>
</html>
```

上記スクリプトをいくつかのWebブラウザで表示させてみると、次のようになります。

▼IE6での表示



▼NN6での表示



これらの取得できる情報を使って、Webブラウザの判定を行うスクリプトはP.108で紹介しています。

■<script>要素で分類する

Webページに JavaScriptを埋め込むには<script>要素が必要となります。この<script>要素の書式は、以下のようになります。

```
<script language="スクリプト名">
```

スクリプト名のところには「JavaScript1.2」や「JavaScript」といった文字列を記述します。この「JavaScript」の後ろに付いたバージョン番号によって、その<script>要素の内容を実行するかどうかが判断されます。

なお、HTML4.01（http://www.w3.org/TR/html4/）の規格では、language属性は非推奨となっています。推奨されているのは、type属性を用いた記述です。

```
<script type="text/javascript">
```

なお、type属性に与えられる値には「text/javascript1.3」などと記述してもかまわないのですが、現在この記述でJavaScriptバージョンの判定が正しく行えるのはIE5以上だけです。ですから、現状では<script>要素によりスクリプトを分割したい場合にはlanguage属性を使わざるを得ません。<script>要素への記述によりスクリプトが動くか動かないかは、次の表のようになっています。「○」が入っているのが動くものです。

<script>要素への記述	IE6	IE5	IE4	NN6	NN4
language="javascript1.5"				○	
language="javascript1.3"	○	○		○	○
language="javascript1.2"	○	○	○	○	○
language="javascript1.1"	○	○	○	○	○
language="javascript"	○	○	○	○	○
type="text/javascript1.5"					
type="text/javascript1.3"	○	○			
type="text/javascript1.2"	○	○	○		
type="text/javascript1.1"	○	○	○		
type="text/javascript"	○	○	○	○	○
language="Jscript"	○	○	○		
type="text/Jscript"	○	○	○		

ここまで解説してくればもうわかると思いますが、JavaScriptの処理をバージョンにより分けたい場合には、<script>要素の書き方次第ということになります。たとえば、JavaScript1.5に対応しているWebブラウザにだけ特殊な処理をさせたいという場合には、以下のような記述となります。

```
<script language="javascript1.5">
   JavaScript1.5に対応しているWebブラウザのための処理
</script>
<script language="javascript1.3">
   JavaScript1.3に対応しているWebブラウザのための処理
   //JavaScript1.5に対応しているWebブラウザもこの処理を行う
</script>
```

ただし、JavaScriptの新しいバージョンに対応しているWebブラウザはすべての処理を行ってしまうため、この方法が適用できる状況はかなり限定されると思います。

■DOMの違いに注目する

Webブラウザの判別の中でも、バージョンやプラットフォームは気にせずにIEとNNの分割だけを行うならば、DOMの違いを利用するのがもっとも簡単な方法です。ただ、NN4とNN6は非常に大きな違いがあるため、最近ではIE5以上とNN6は同じ処理で、NN4だけを別の処理にする場合が多くなってきています。

そのため、3つのオブジェクトを用いてそれぞれのWebブラウザの判別をします。

本書ではIEでしか動作しない"document.all"オブジェクト、IE5以上とNN6でしか動作しない"document.getElementById"オブジェクト、NN4でしか動作しない"document.layers"オブジェクトを用います。それぞれのオブジェクトと、Webブラウザの動作の対応は以下のようになっています。

オブジェクト	IE4	IE5	IE6	NN4	NN6
document.all	○	○	○		
document.getElementById		○	○		○
document.layers				○	

では、動作するかどうかというのはどのように調べるのでしょうか？　実は非常に簡単で、if文でオブジェクトを条件式に使えばよいのです。条件式に記述されたオブジェクトが動作すればtrueが返ってくるので、if文は真となります。

実際にオブジェクトを使って処理を分けるスクリプトは、以下のようになります。

```
<html>
<title>browser separate by DOM</title>
<body>
<h3>browser check by DOM</h3>
<hr />
<script type="text/javascript">
<!--
if(document.getElementById){
   document.write("DOM OK (IE5+, NN6+)<br />");
}

if(document.all){
   document.write("IE OK <br />");
}

if(document.layers){
   document.write("NN4 OK");
}
```

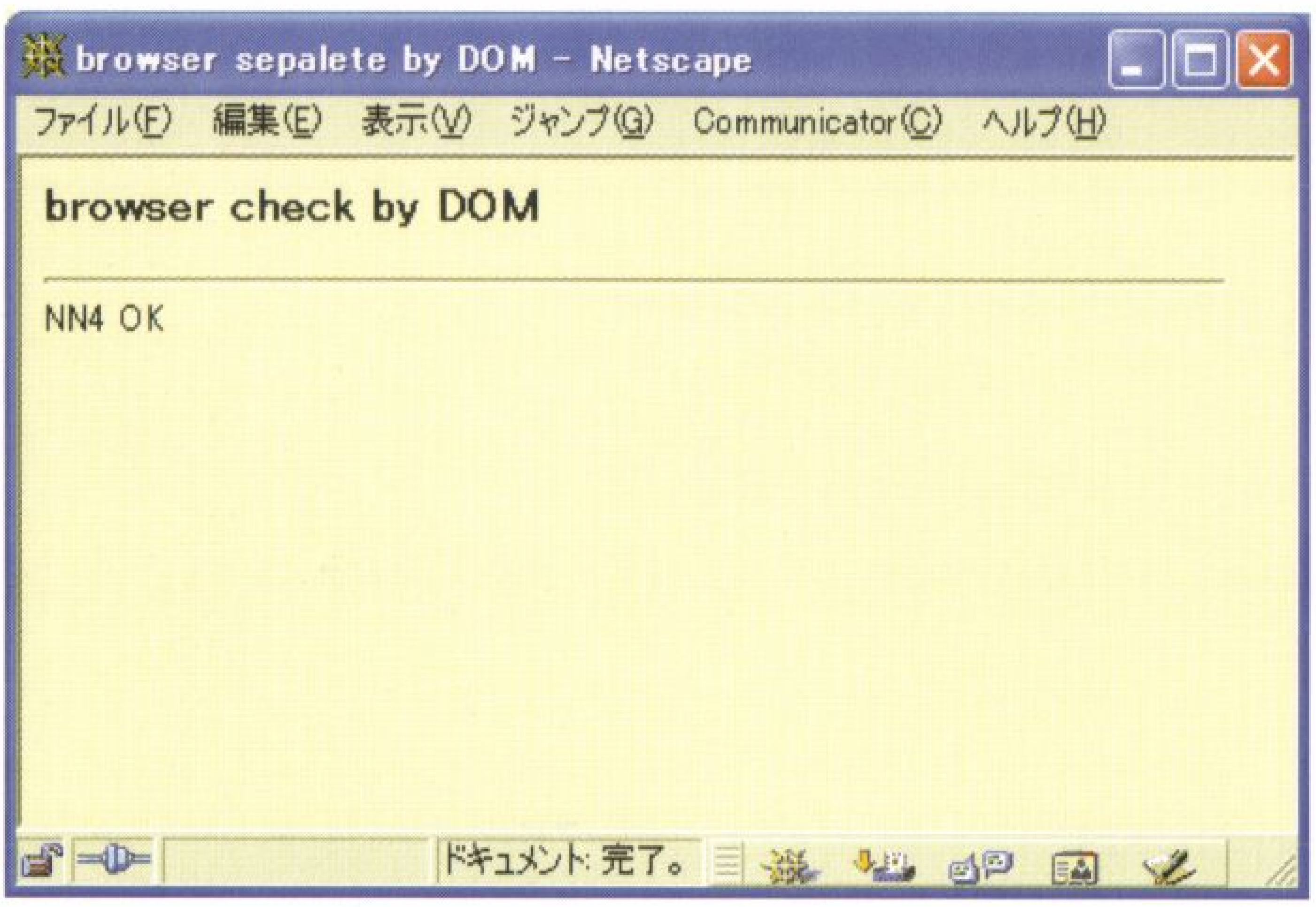
```
//-->
</script>
</body>
</html>
```

このスクリプトを実際に動作させると、以下のようになります。

▼IE6での表示



▼NN4での表示



このように、たったこれだけの条件式でWebブラウザの判定ができます。ただし、OSによる違いを検出してWebページを変えたいといった場合や、細かなバージョンの違いにより処理を分離したいといった場合には、navigatorオブジェクトからWebブラウザ情報を格納しているプロパティを使って判定する必要があります。

JavaScript
Handbook
3rd edition

3-1 WIN MAC

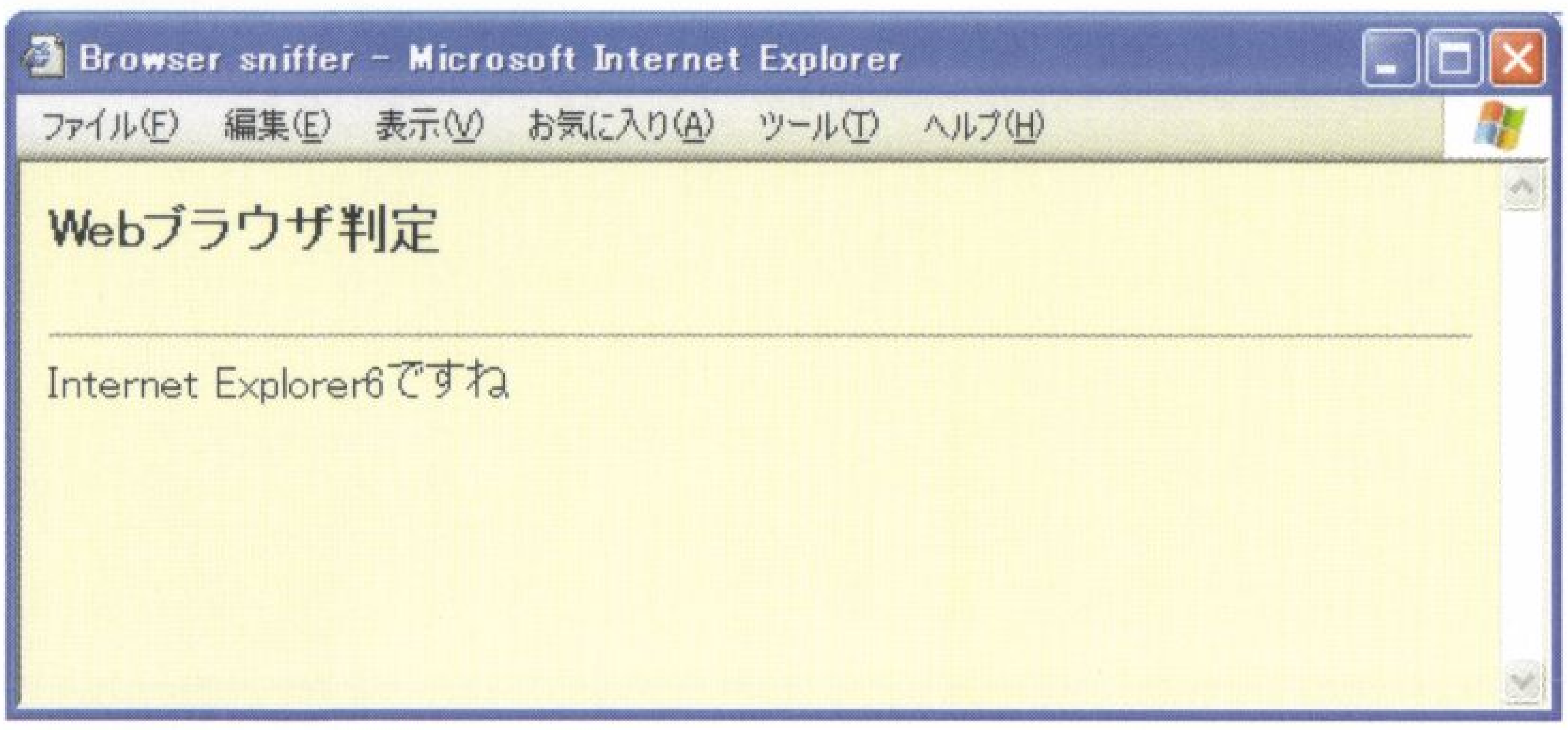
ユーザーが使用しているWebブラウザを判定する

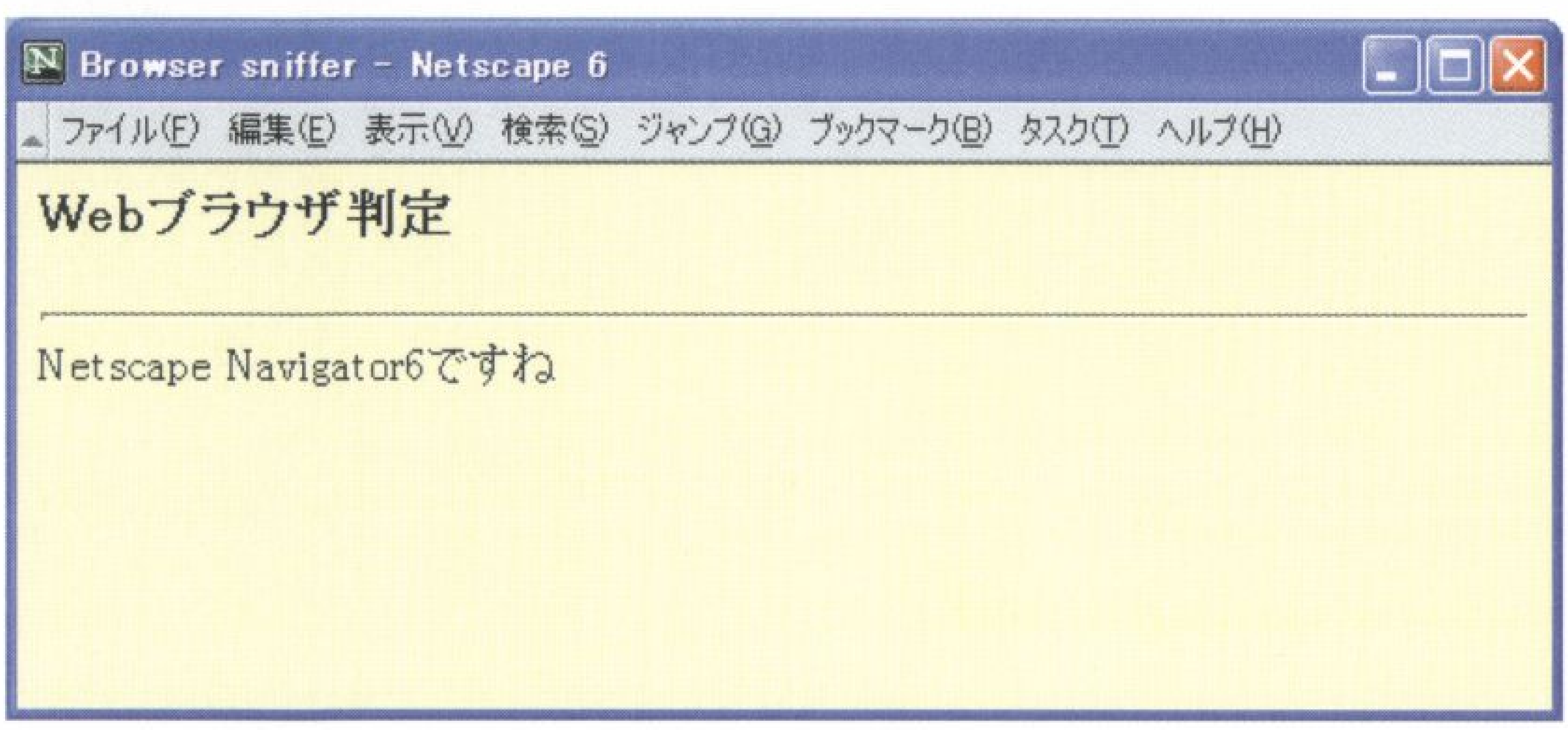
`navigator.userAgent.indexOf('参照する文字列')>=0`

Webブラウザの違いによって、思いもしないWebページの見え方になってしまうことがあります。特にDHTMLやCSS、さらには本書の対象であるJavaScriptも、Webブラウザの種類やそのバージョンによって動作が異なります。

これらの問題を解決するにはどうしたらよいでしょうか？　1つには、どのWebブラウザにも共通な部分だけを使ってWebページを制作するという方法があります。しかし、これではできることが非常に限られてしまいます。そこで、ユーザーが使用しているWebブラウザを判定し、それぞれに最適化したWebページへユーザーを移動させる方法があります。

DISPLAY





SOURCE

sniffer.html

```
<html>
<title>sniffer</title>
<script language="javascript">
<!--
   var ua, mode;
   ua = navigator.userAgent;  ―①
   if(ua.indexOf("MSN 6", 0) >= 0){
      mode = 2;
   }
   else if(ua.indexOf("MSIE 6",0) >= 0){
      mode = 10;
   }
   else if(ua.indexOf("MSIE 5",0) >= 0){
      mode = 11;
   }
   else if(ua.indexOf("MSIE 4",0) >= 0){  ―②
      mode = 12;
   }
   else if(ua.indexOf("Netscape6",0) >= 0){
      mode = 20;
   }
   else if(ua.indexOf("Mozilla/4",0) >= 0){
      mode = 21;
   }
   else{
      mode = 99;
   }
//-->
</script>
<body bgcolor="#fff8dc">
<h3>Webブラウザ判定</h3>
<hr />
<script language="javascript">
<!--
   if(mode==2){
      document.write("MSN Explorer6ですね¥n");
   }
   else if(mode==10){
```

▼

```
        document.write("Internet Explorer6ですね¥n");
    }
    else if(mode==11){
        document.write("Internet Explorer5ですね¥n");
    }
    else if(mode==12){
        document.write("Internet Explorer4ですね¥n");
    }
    else if(mode==20){
        document.write("Netscape Navigator6ですね¥n");
    }
    else if(mode==21){
        document.write("Netscape Navigator4ですね¥n");
    }
    else if(mode==99){
        document.write("それ以外のWebブラウザですね¥n");
    }
//-->
</script>
</body>
</html>
```

DETAILS

Webブラウザは、navigatorオブジェクトにさまざまな情報を保持しており、それを利用してWebブラウザを判定することが可能です。具体的にはuserAgentプロパティに入っているWebブラウザ情報を取り出し、その情報を使ってWebページを訪れたユーザーのWebブラウザを判定します。userAgentプロパティにはWebブラウザにより固有の文字列が格納されています。

各Webブラウザに格納されている情報は、次ページの表のようになっています。このuserAgentプロパティから、Webブラウザを特定するのに必要な情報を取り出します。

Webブラウザ	userAgentプロパティに格納されている情報
MSN Explorer6	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.2914; MSN 6.1; MSNbMSFT; MSNmja-jp; MSNcOTH)
IE6	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.2914)
IE5	Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
IE4	Mozilla/4.0 (compatible; MSIE 4.01; WinNT 5.0)
NN6	Mozilla/5.0 (Windows; U; Windows NT 5.0; ja-JP; rv:0.9.2) Gecko/20010726 Netscape6/6.2
NN4	Mozilla/4.78 [ja] (Windows NT 5.0; U)

たとえば、WebブラウザがIE6かどうかを判定するためには、「MSIE 6」という文字列がuserAgentプロパティ中にあるかどうかを確認します。

```
navigator.userAgent.indexOf('MSIE 6')>=0
```

これは「navigator.userAgent」でWebブラウザの情報を参照し、indexOfメソッドで文字列内に「MSIE 6」という文字列があるかどうか調べています。ない場合、indexOfメソッドは「-1」を返し、条件はfalseとなります。

サンプルでは①の個所で、userAgentプロパティの情報を取得しています。

```
ua=navigator.userAgent;
```

変数に代入しているのは、毎回「navigator.userAgent」と書いているとコードが長く、読みづらくなるからです。

次に、識別文字列について考えてみます。このサンプルでは、次のWebブラウザに関して識別を行っています。

Webブラウザ	識別文字列
MSN Explorer6	MSN 6
IE6	MSIE 6
IE5	MSIE 5
IE4	MSIE 4
NN6	Netscape6
NN4	Mozilla/4

それぞれのブラウザを、識別文字列で判定していますが、userAgentの情報の表にあったように、すべてのWebブラウザに「Mozilla」という文字列が存在し、MSN Explorer6、IE6、IE5、IE4、NN4には、「Mozilla/4」という文字列が格納されている点に注意してください。そのため、「Mozilla/4」を最初に持ってくると、MSN Explorer、IE6、IE5、IE4とNN4が区別できなくなります。

サンプルの②の部分では、まず、MSN Explorer6を他のWebブラウザと区別できる「MSN 6」を識別文字列に使っています。その後で「MSIE 6」「MSIE 5」「MSIE 4」を識別文字列に使い判定しています。ここまでで、MSN ExplorerとすべてのIEの判別ができています。次にNN6とNN4の判別です。NN6には「Netscape6」という文字列がuserAgentプロパティに格納されているので、これを識別に使っています。

判定ができたら、それぞれの場所で変数modeに数字をセットしています。なお、変数modeの値は次のように定義しています。

Webブラウザ	変数modeの定義
MSN Explorer6	2
IE6	10
IE5	11
IE4	12
NN6	20
NN4	21
その他	99

変数modeの値がどのWebブラウザかを表しており、これを使って処理を分岐させます。今回は<body>～</body>内のスクリプトでWebブラウザごとに表示させる文字列を変えています。

T I P S

実際に実装するときには、「Webブラウザ判定の処理」と「Webブラウザごとの処理」をひとまとめにして記述しないほうがよいでしょう。なぜなら、「Webブラウザごとの処理」はWebページ内のさまざまな場所で使う可能性があり、ひとまとめにして書いてしまうと、そのたびごとに記述する必要があるためです。

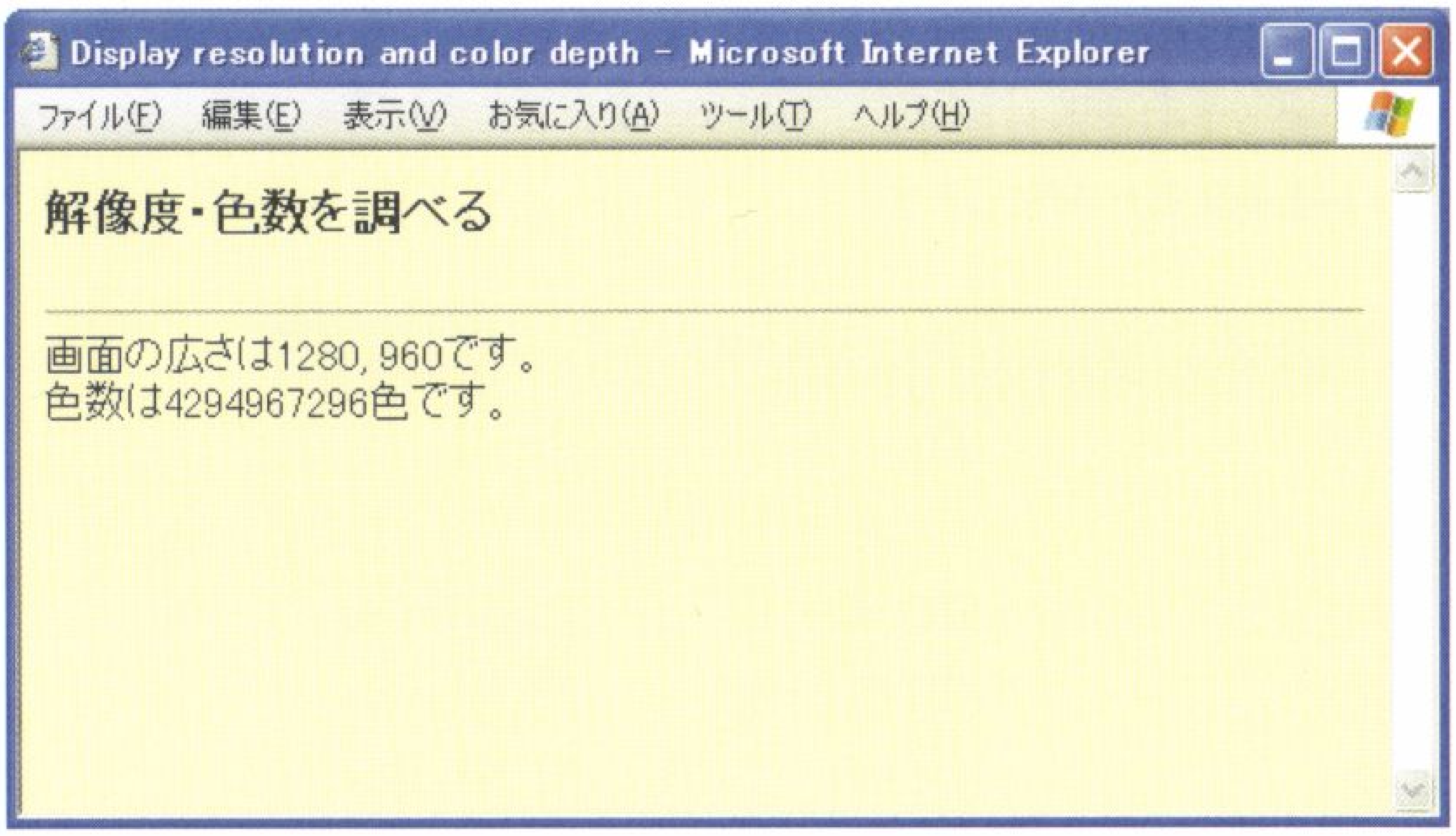
サンプルのスクリプトでは、<body>タグの前で「Webブラウザ判定の処理」を、<body>～</body>内で「Webブラウザごとの処理」を行っています。

解像度、色数のチェックを行う

```
screen.width
screen.height
screen.colorDepth
```


Webブラウザは、PC環境に依存するため、ユーザー環境はさまざまです。そのため、画面が広いことを前提にしているコンテンツや、画面の色数が多いことを前提にしているコンテンツは、画面の狭いPCや表示できる色数の少ないPCでは、正しく表示されない可能性があります。そんなときには、事前にユーザーのPC環境を調べておけば、対処できることがあります。

D I S P L A Y



S O U R C E

displayInfo.html

```
<html>
<title>DisplayInfo</title>
<body bgcolor="#fff8dc">
<h3>解像度・色数を調べる</h3>
<hr />
<script language="javascript">
<!--
   var displayWidth = screen.width;
   var displayHeight = screen.height;
   var cd = screen.colorDepth;
```

```
    document.write("画面の広さは" + displayWidth + ",
⇒       " + displayHeight + "です。<br />")
    document.write("色数は" + Math.pow(2,cd) + "色です。");
//-->
</script>
</body>
</html>
```

BROWSER

DETAILS

このスクリプトでは、screenオブジェクトのwidthプロパティ、heightプロパティ、colorDepthプロパティを用いてユーザー環境について調べています。

widthプロパティはディスプレイの幅を、heightプロパティはディスプレイの高さの情報を格納しています。このとき数値の単位はピクセル(pixel)です。

colorDepthプロパティは、ディスプレイの色を表示するために用いるカラーパレットのビット数を返します。これは1ピクセルの色を表現するのに、どれだけのビットを使えるかを表します。

返される値(ビット数)	色数	備考
1ビット	2色	白・黒
8ビット	256色	
16ビット	65536色	HighColor
24ビット	16777216色	TrueColor
32ビット	4294967296色	TrueColor

このサンプルでは、返されるビットの値を色数に変換するために、次のように記述しています。

```
Math.pow(2,cd)
```

この記述では、返される値に対して2のべき乗を行い、ビット数から色数に変換しています。

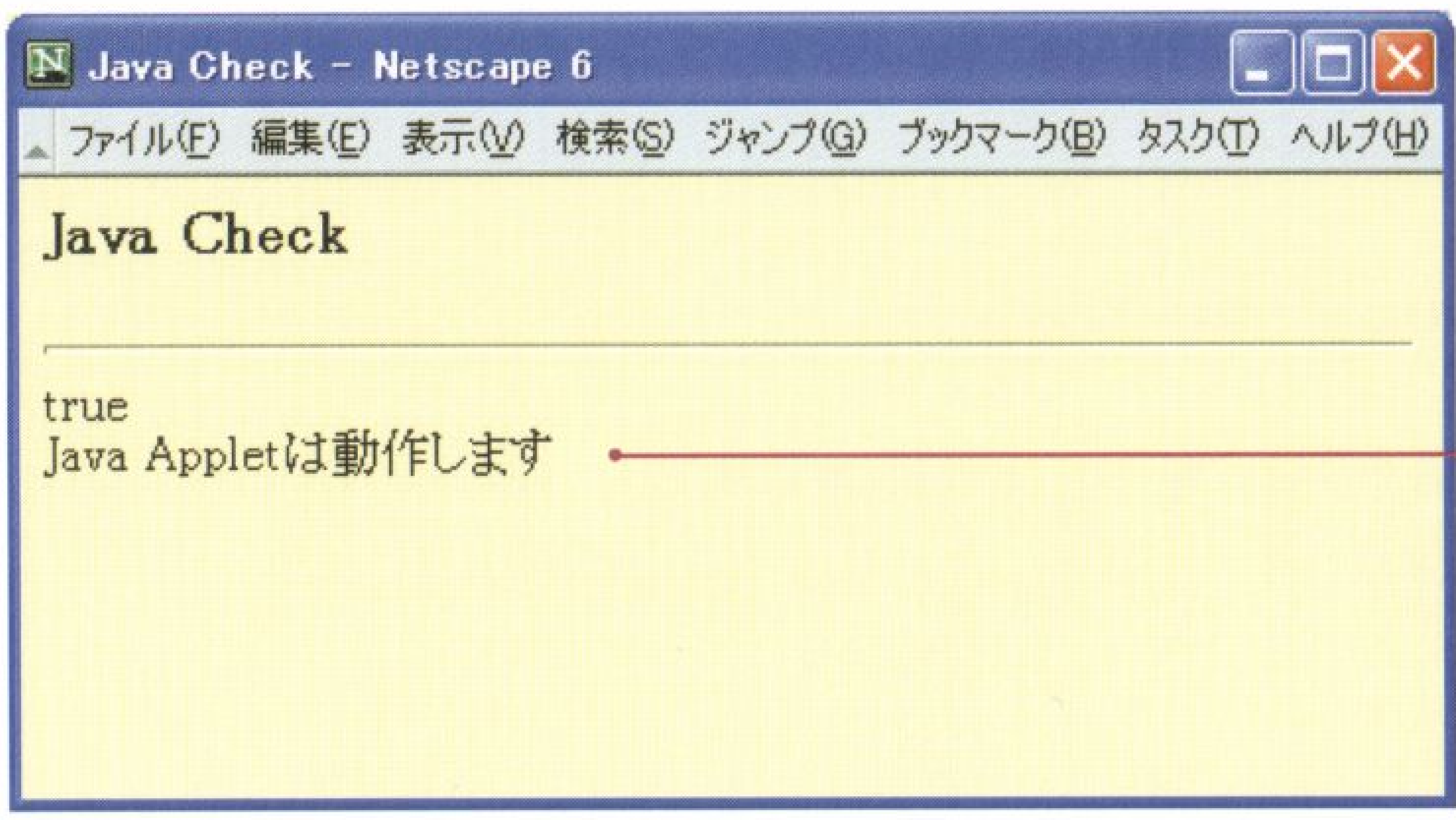
3-3 WIN 4↑ 4↑ MAC 4↑ 4↑

Javaが有効かどうかをチェックする

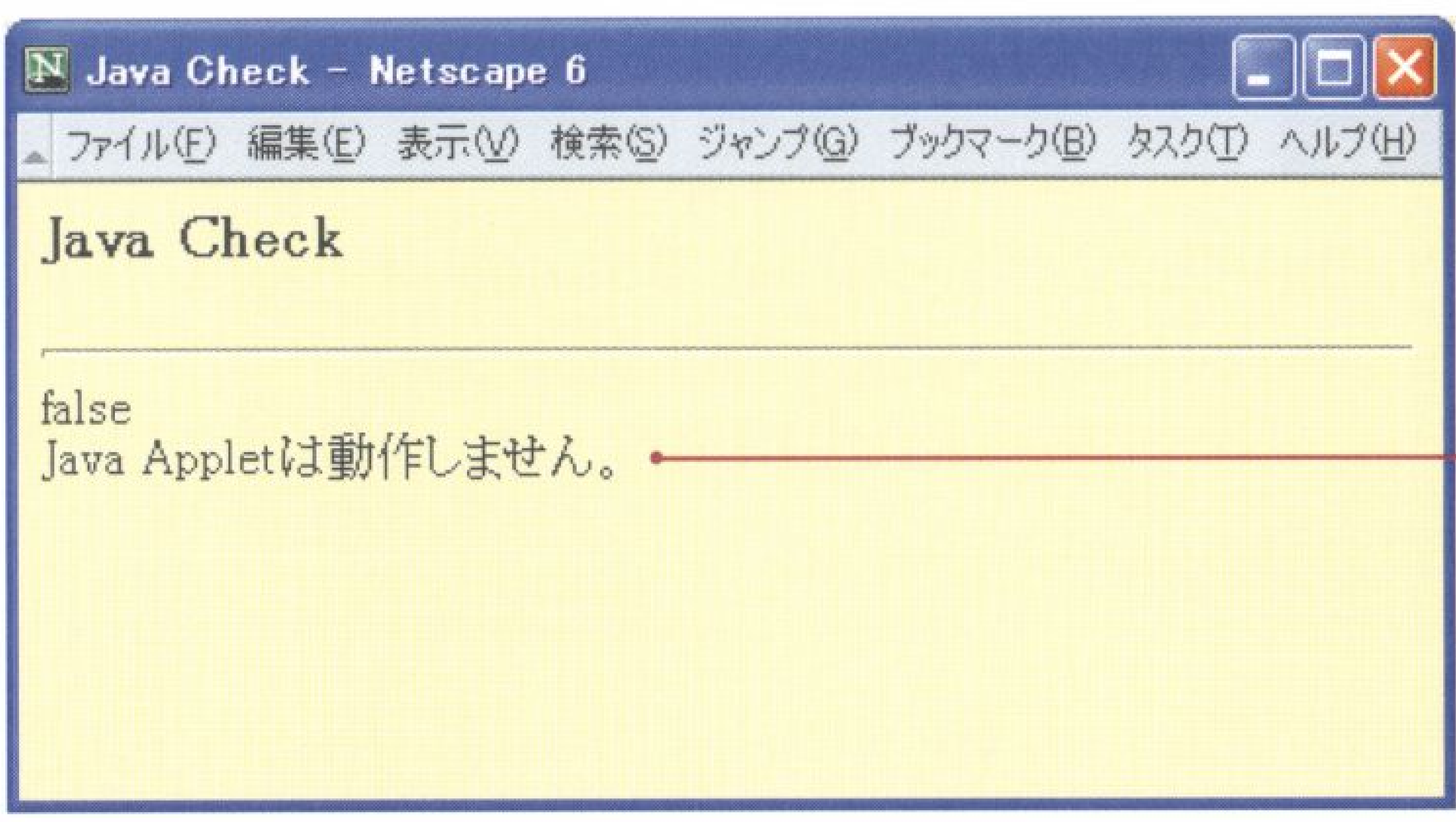
navigator.javaEnabled()

Javaアプレットが動作可能かどうかを調べるスクリプトです。ただ、最近ではJavaアプレットをインターネットのWebページ上で動作させることはかなり少なくなってきています。また、Windows XPやIE6は標準ではJava VM（Microsoft VM）がバンドルされていません。そのため、すべてのWebブラウザで、必ずしもJavaが動作するわけではありません。また、Webブラウザ側でJavaアプレットが動作しないように設定してある場合もあります。このようなときのために、Javaアプレットが動作可能であるかどうか調べる必要があります。

DISPLAY



Javaアプレットが有効のときの表示



Javaアプレットが無効のときの表示

SOURCE

java_check.html

```
<html>
<title>Java Check</title>
<script language="javascript">
<!--
   function javaCheck(){
      if(navigator.javaEnabled()){
         document.write("Java Appletは動作します。");
      }
      else{
         document.write("Java Appletは動作しません。");
      }
   }
//-->
</script>
<body bgcolor="#fff8dc">
<h3>Java Check</h3>
<hr />
<script language="javascript">
<!--
   document.write(navigator.javaEnabled() + "<br />");
   javaCheck();
//-->
</script>
</body>
</html>
```

DETAILS

Javaアプレットが動作可能かどうかを調べるにはnavigatorオブジェクトのjavaEnabledメソッドを使います。書式は以下のようになります。

```
navigator.javaEnabled()
```

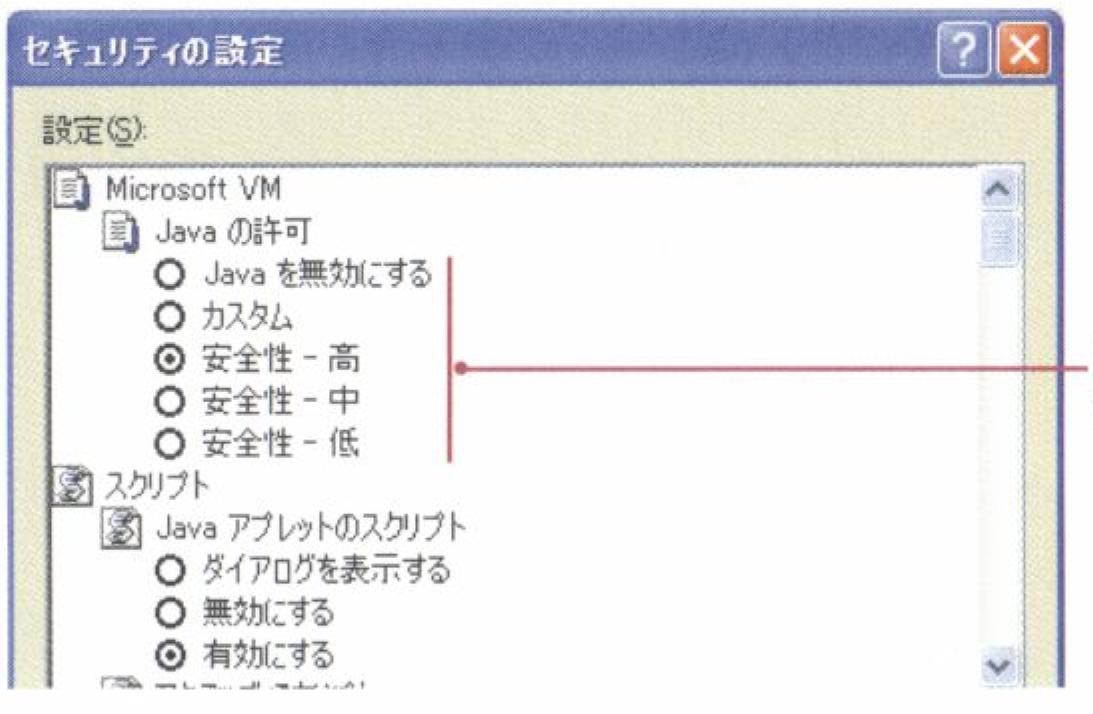
このメソッドはJavaアプレットが動作するならば「true」を、Javaアプレットが動作しないときには「false」を返します。trueならば「Javaアプレットは動作します。」、falseならば「Java Appletは動作しません。」と表示するようにしてい

ます。このメソッドを用いてJavaアプレットの動作の可否を調べ、動作しないときには、Javaアプレットを動作可能にするにはどうしたらよいかを表示するようにしておくとよいでしょう。

T I P S

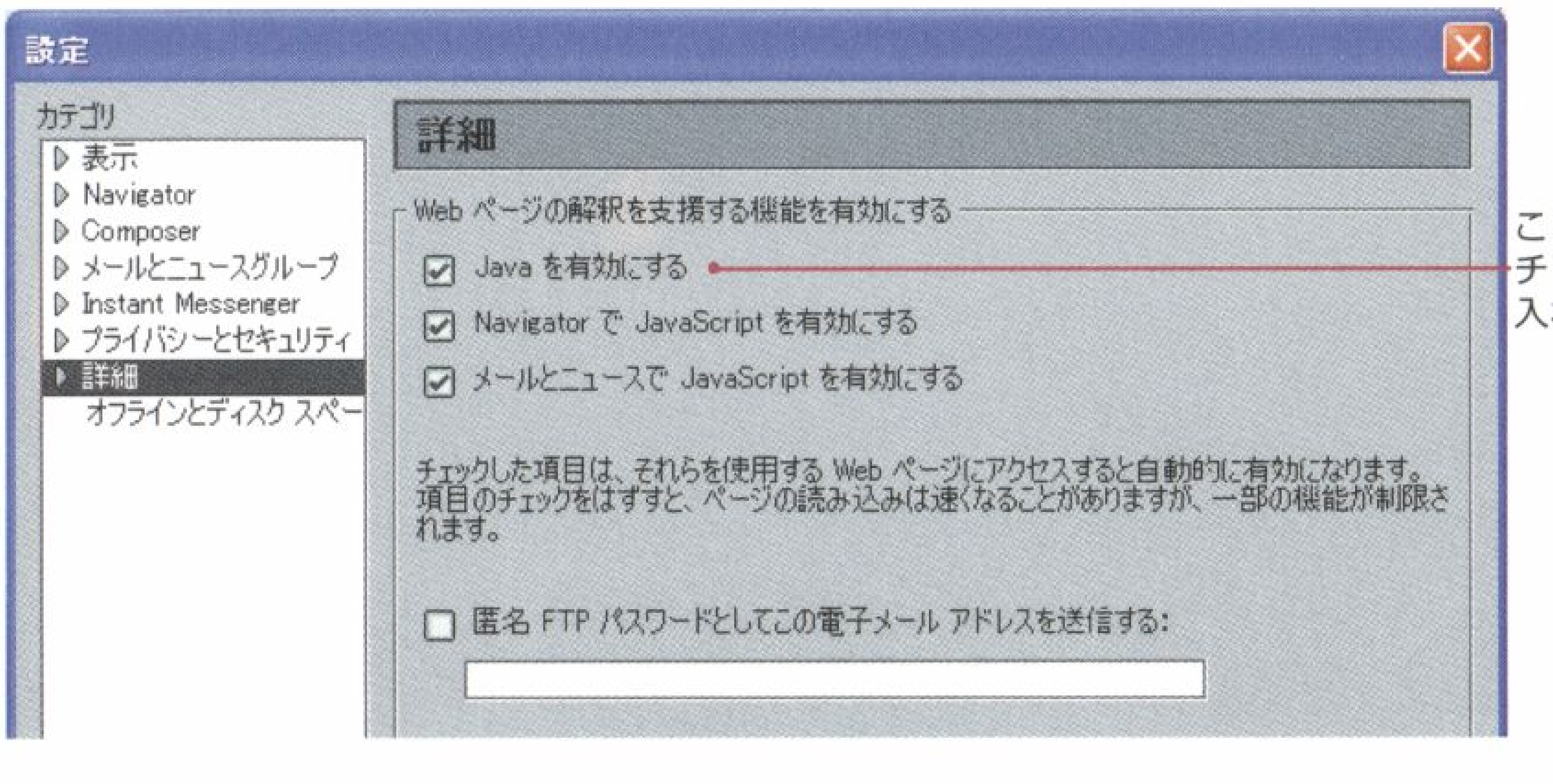
WindowsのIEとNNで、Javaアプレットを有効にする方法を紹介しておきます。IE6は、標準でMicrosoft VM(Java VM)を搭載していませんが、必要なときには、Microsoft社のWebページからダウンロードするようになっています。

IE5とIE6の場合には、メニューバーで【ツール】→【インターネットオプション】を選択し、[インターネットオプション]ダイアログボックスの[セキュリティ]パネルを開きます。パネル下の「レベルのカスタマイズ」ボタンをクリックして開く[セキュリティの設定]パネルで、Javaの設定を行います。



「Javaを無効にする」以外を選択すると有効になる

NN4やNN6の場合には、メニューバーで【編集】→【設定】を選択して、[設定]パネルを開きます。「カテゴリ」の「詳細」を選択したパネルで、「Javaを有効にする」にチェックを入れると、有効になります。



この項目にチェックを入れておく

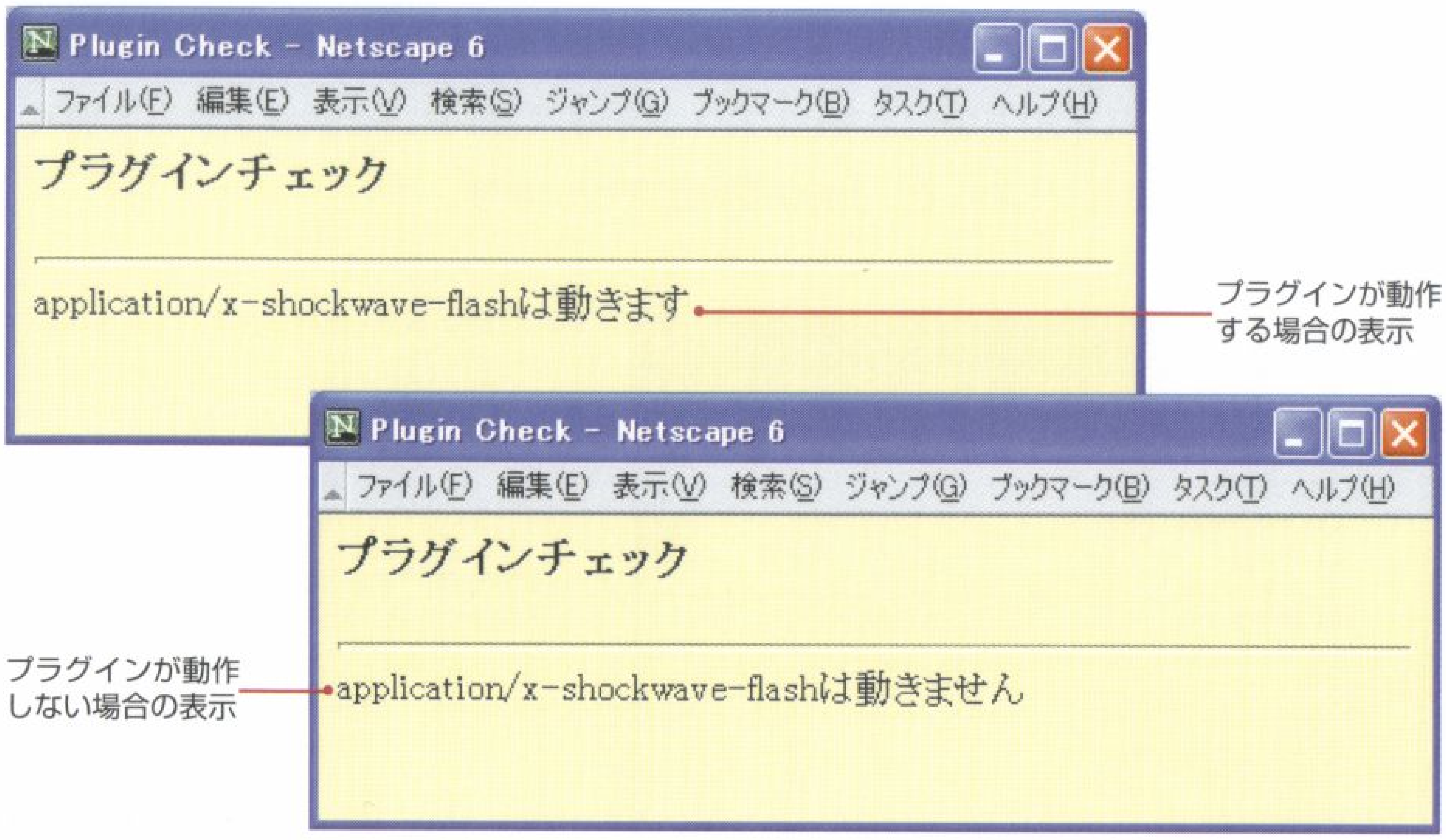
3-4 WIN MAC

必要なプラグインが動作するかどうかチェック

navigator.mimeTypes[プラグインのMIMEタイプ].
enabledPlugin

このスクリプトはページに埋め込まれているプラグインが動作するかどうかチェックするものです。IEはこのスクリプトで使っている、navigator.mimeTypesをサポートしていないため動作しません。

DISPLAY



SOURCE pugincheck.html

```
<html>
<title>Plugin Check</title>
<body bgcolor="#fff8dc">
<h3>プラグインチェック</h3>
<hr />
<script language="javascript">
<!--
var TargetPlugin="application/x-shockwave-flash";  ①
if(navigator.mimeTypes[TargetPlugin].enabledPlugin){  ②
    document.write(TargetPlugin,"は動きます");}
else{
    document.write(TargetPlugin,"は動きません");
}
```

```
//-->
</script>
</body>
</html>
```

DETAILS

スクリプトの①の部分で確認したいプラグインを設定しています。ここでは、Macromedia社のShockwave Flashが動作するかどうかをチェックするために、Shockwave FlashのMIMEタイプである「application/x-shockwave-flash」を設定しています。実際にプラグインが動作するかどうかは、enabledPluginプロパティを用います（②の箇所）。enabledPluginプロパティの書式は、次のようになります。

```
navigator.mimeTypes[プラグインのMIMEタイプ].enabledPlugin
```

enabledPluginプロパティには、ターゲットとするプラグインが動作するならば「true」が、動作しない場合には「false」が格納されています。それをif文で判定して分岐させています。ここでは、Shockwave Flashをチェックしましたが、これ以外の主要なプラグインのMIMEタイプは、次の表のようになります。

プラグイン	MIMEタイプ
Macromedia Shockwave for Director	application/x-director
Macromedia Shockwave for FreeHand	application/x-freehand
Apple QuickTime	video/quicktime
Adobe PDF	application/pdf

TIPS

IEが「navigator.mimeTypes」をサポートしていないのは、プラグインに対しての考え方に違いがあるからです。Microsoft社ではWebブラウザに新しい機能を組み込む場合、ActiveXコントロールで提供しようとしています。ActiveXコントロールは、その機能がインストールされていない場合、自動的にインストールまで行うので、ActiveXコントロールが自動的に組み込めるようにWebページが記述してあれば問題ないのです。次ページにshockwave FlashのActiveXコントロールを自動的に組み込むサンプルを挙げておきます。埋め込みには<object>タグを用い、Shockwave Flashのファイル名は「index.swf」

とします。

plugin_activex.html

```
<html>
<title>プラグインチェック</title>
<body>
<h3>プラグインチェック-&lt;object&gt;タグ版</h3>
<hr />
<script language="javascript">
<!--
var ua=navigator.userAgent;
if(document.layers){
    var TargetPlugin="application/x-shockwave-flash";
    if(navigator.mimeTypes[TargetPlugin].enabledPlugin){
        document.write(TargetPlugin,"は動きます。");}
    else{
        document.write(TargetPlugin,"は動きません。Pluginをダウン
⇒           ロードしてください。");
    }
}
//-->
</script>
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-4445535400
⇒   00" codebase="http://download.macromedia.com/pub/
⇒   shockwave/cabs/flash/swflash.cab#version=5,0,0,0"
⇒   width="300" height="120" name="sw" id="sw">
    <param name="Movie" value="index.swf" />
    <param name="quality" value="best" />
    <param name="play" value="true" />
    <embed src="flash.swf" pluginspage="/shockwave/
⇒       download/index.cgi?P1_Prod_Version=
⇒       ShockwaveFlash"type="application/x-shockwave-flash"
⇒       width="300" height="120" loop="true" quality="best"
⇒       bgcolor="#000000">
    </embed>
</object>
</body>
</html>
```

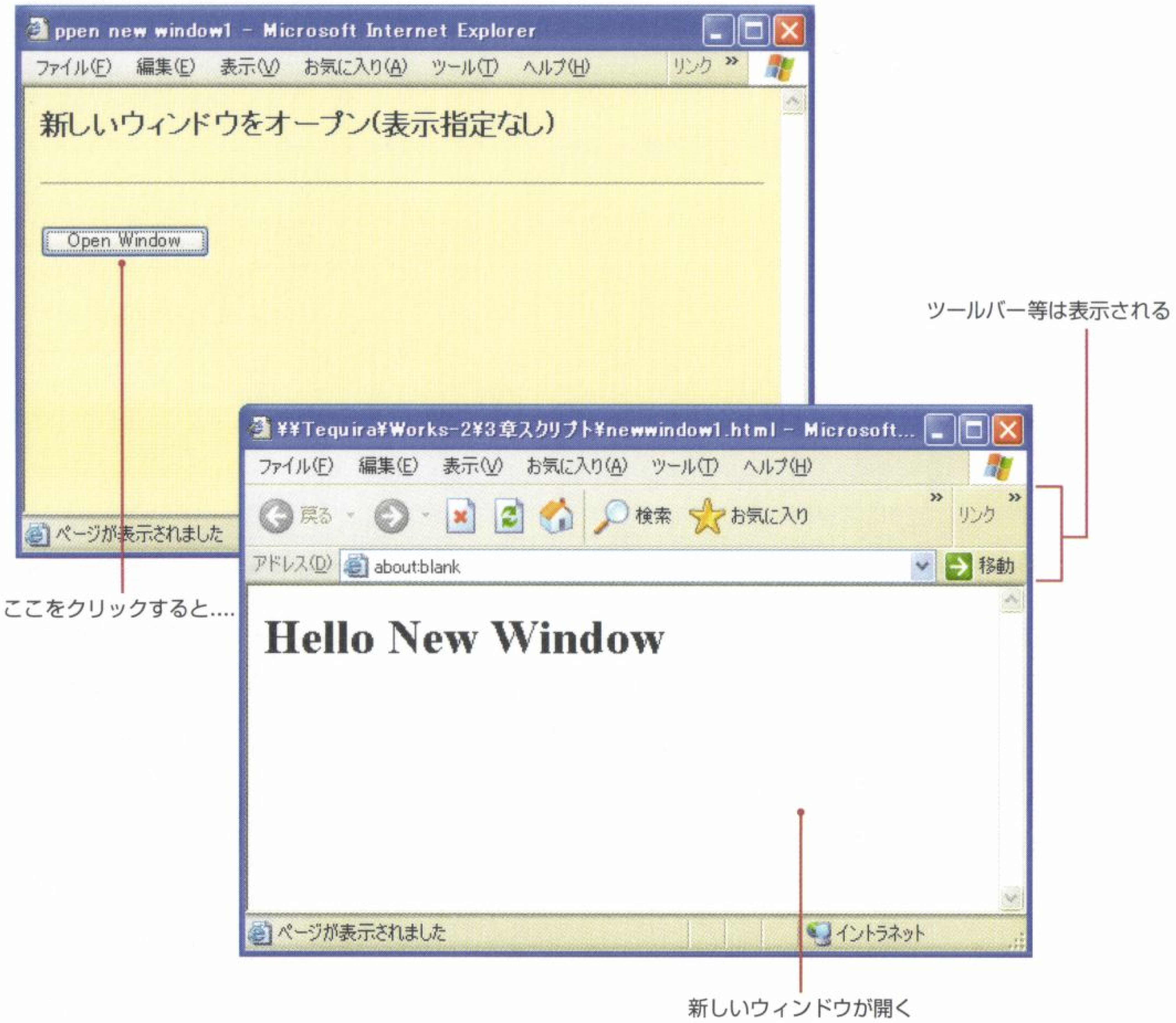
4-1 WIN 4↑ N 4↑ MAC 4↑ N 4↑

新しいウィンドウを開く

```
window.open('URL','ウィンドウ名','スタイル');
windowオブジェクト名.document.write('表示するテキスト');
windowオブジェクト名.document.close();
```

ウィンドウを新たに生成するスクリプトです。生成するウィンドウには、識別するための名前を付けています。このスクリプトでは、フォームのボタンをクリックすると名前が付いたウィンドウが現れます。

DISPLAY



SOURCE

newwindow1.html

```
<html>
<title>open new window1</title>
<script language="javascript">
<!--
function openWin(){
    var win1 = window.open('','newwin1',''); ——②
    win1.document.write('<h1>Hello New Window</h1>'); ——③
    win1.document.close(); ——④
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>新しいウィンドウをオープン(表示指定なし)</h3>
<hr />
<form>
<input type="button" value="Open Window" onclick="openWin()" />
</form>                                              ——①
</body>
</html>
```

DETAILS

Webページ上に配置されたフォームボタンをクリックすると、<input>タグのonclickイベントハンドラによってopenWin関数が呼び出されます(①の箇所)。openWin関数では、ウィンドウを新たに開いて、その新しいウィンドウに「Hello New Window」と書き込んでいます。新たなウィンドウを開くにはwindowオブジェクトのopenメソッドを用います。このメソッドは、以下の形式で使用します。

```
window.open('URL','ウィンドウ名','スタイル');
```

サンプルではウィンドウ名のみを指定し、それ以外のパラメータについては省略しています(②の箇所)。このウィンドウ名は、それぞれのウィンドウにリンクを張るときに、ターゲットとなるウィンドウを指定するために使います。また、新しく生成されたウィンドウは、このサンプル中では変数「win1」という名前で識別されます。スタイルに関してはP.128を参照してください。

次に③で、新たに生成したwin1ウィンドウに文字列を表示させています。このようにウィンドウに文字列を表示させたい場合は、次のような書式によりwindowオブジェクトを指定して、writeメソッドを使います。

```
windowオブジェクト名.document.write('表示するテキスト');
```

'表示するテキスト'にはタグを含めることも可能ですし、上記の書式を並べることにより複数行の文字列を表示させることもできます。

文字列を書き込んだ後、④のようにdocumentオブジェクトを閉じる必要があります。

```
windowオブジェクト名.document.close();
```

このcloseメソッドを忘れると、スクロールバーが表示されない、データ読み込み状態のままになってしまうなどの問題が起こることがあります。

TIPS

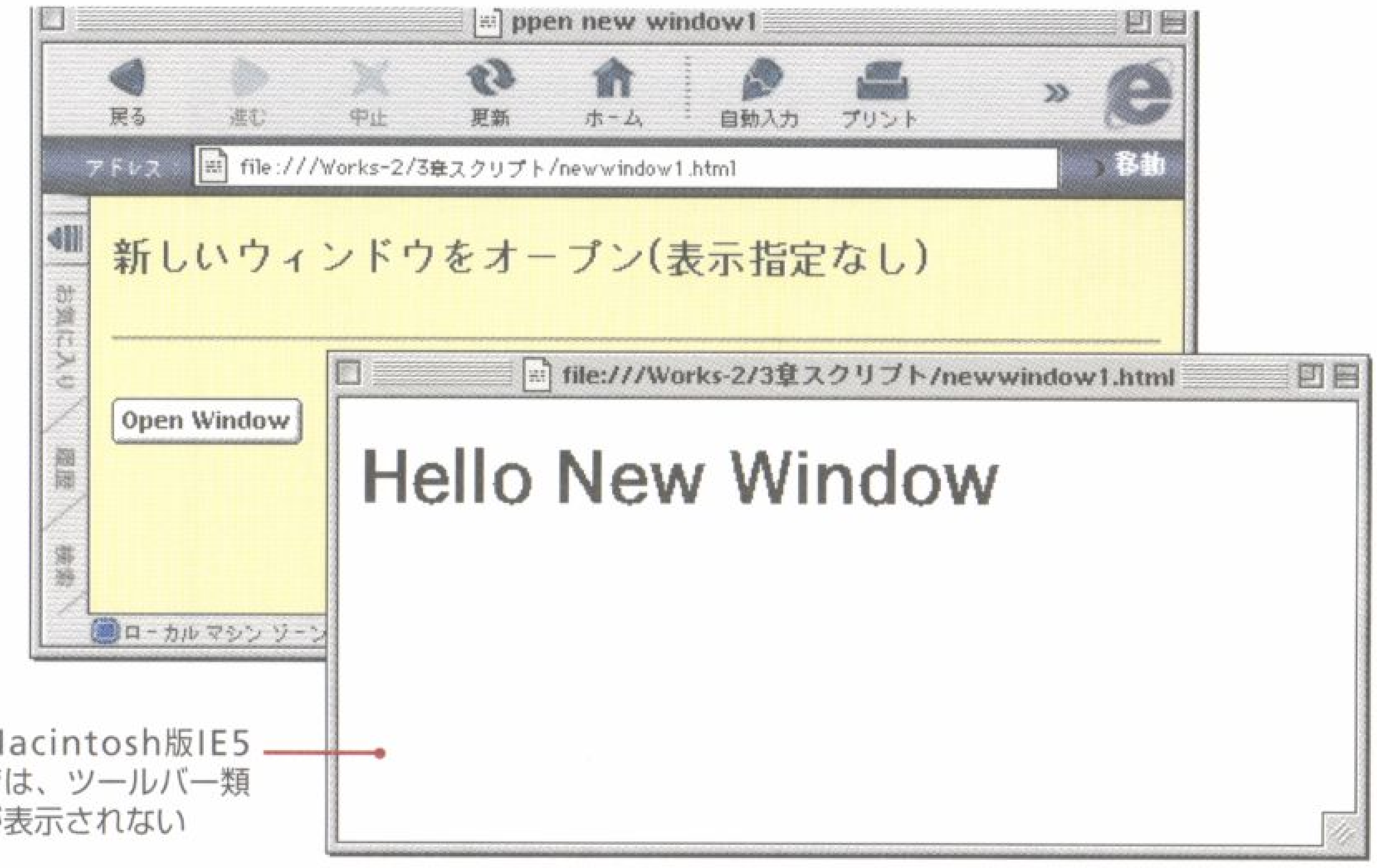
このスクリプトでは、スタイルを指定しないでウィンドウを開いていますが、開かれるウィンドウの初期値はWebブラウザによって、微妙に異なっているようです。

まず「DISPLAY」の画面を見てもらえばわかるように、Windows版のIE6では新しく開かれるウィンドウに必ずすべてのツールバーとステータスバーが表示されます。これに対してNN6では、元のウィンドウのスタイルを引き継ぐようになっているようです。

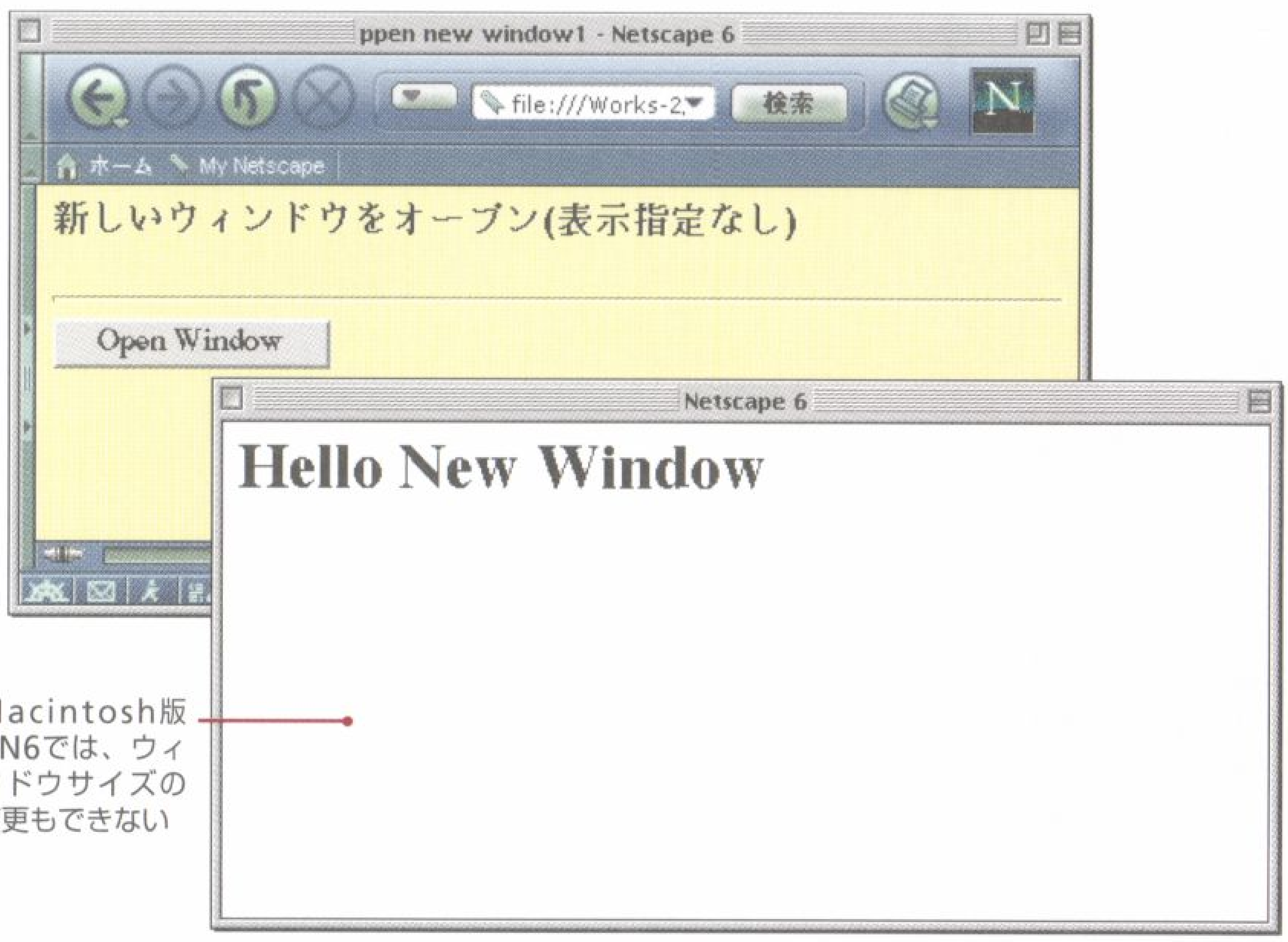
一方、Macintosh版のWebブラウザでは、ツールバーの類は一切表示されません。さらにNNでは、スクリプトにより開かれるものは通常のウィンドウとは属性が違うらしく、ツールバーなどを表示させることができないうえ、ウィンドウサイズを変更することもできません。



親ウィンドウのスタイルを引き継ぐ



Macintosh版IE5では、ツールバー類が表示されない



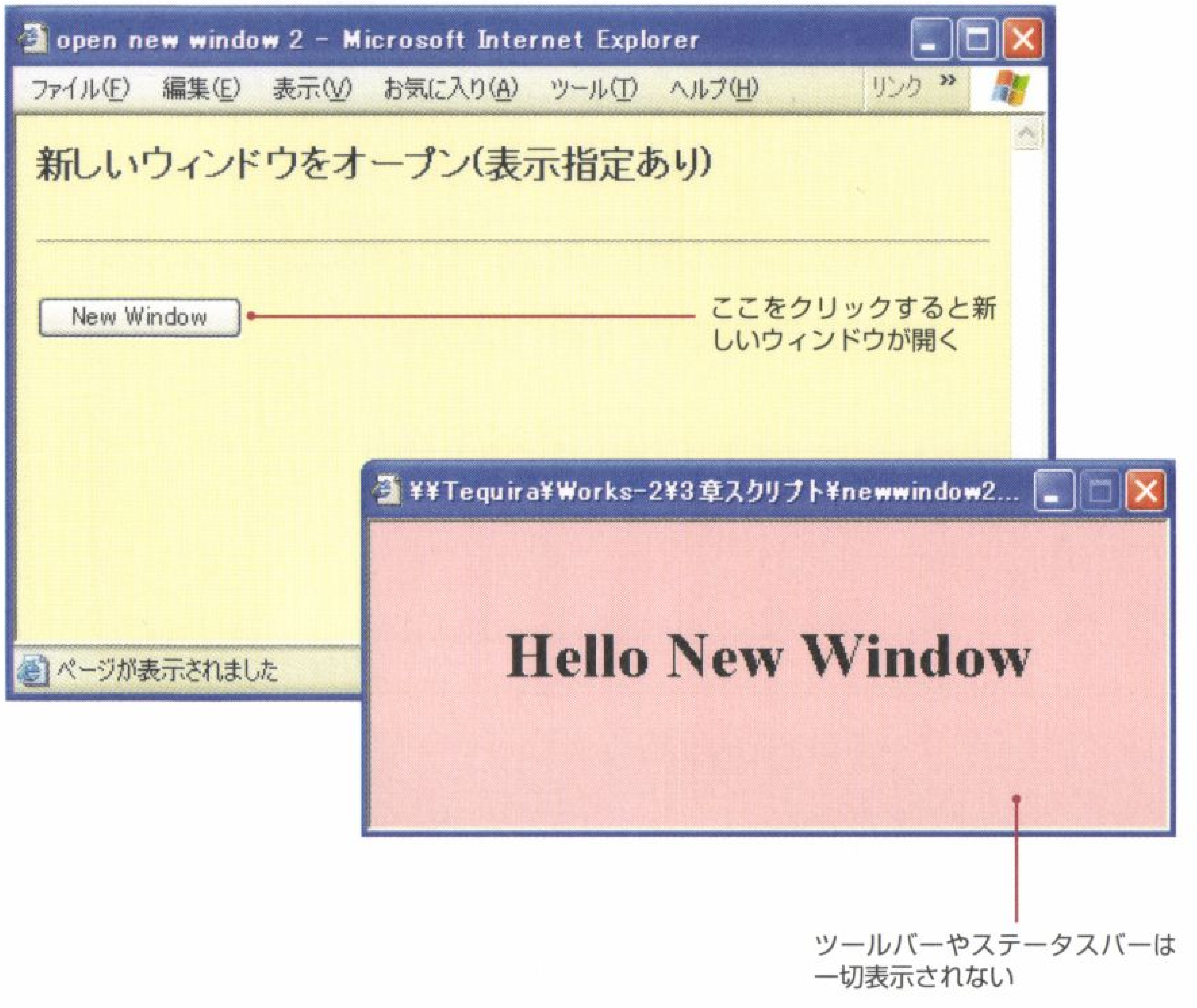
Macintosh版NN6では、ウィンドウサイズの変更もできない

新しいウィンドウを開く(サイズ・スタイル指定)

```
window.open('URL','ウィンドウ名','スタイル');
```

ちょっとした注釈や説明を行うために新しいウィンドウを開くことはとても有効です。特に注釈のような文章は、そのつどリンク先に飛んでも不便です。また、小さいウィンドウで表示したいときに、ツールバーなどのバーが邪魔に感じられたこともあるかと思います。ここでは、さまざまなバーが表示されていない別ウィンドウを出現させてみます。

DISPLAY



SOURCE

newwindow2.html

```
<html>
<title>open new window 2</title>
<script language="javascript">
<!--
function openWin(){
   var win1=window.open('','newwin1','toolbar=no,location       ┐
⇒     =no,status=no,menubar=no,scrollbars=no,                   ├①
⇒     width=400,height=150');                                   ┘
   win1.document.write('<body bgcolor=#FFCCCC>');               ┐
   win1.document.write('<table width="100%" height="100%">');   │
   win1.document.write('<tr align="center">');                  ├②
   win1.document.write('<td><h1>Hello New Window</h1></td>');   │
   win1.document.write('</tr></table>');                        │
   win1.document.write('</body>');                              ┘
   win1.document.close();
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>新しいウィンドウをオープン(表示指定あり)</h3>
<hr />
<form>
   <input type="button" value="New Window" onclick="openWin()"
⇒     />
</form>
</body>
</html>
```

DETAILS

基本的な構造は前項で紹介したものと同じですが、このサンプルでは新しいウィンドウを表示する際のさまざまなステータスを指定しています（①の箇所）。

```
toolbar=no,location=no,status=no,menubar=no,scrollbars=no,
⇒  width=400, height=150
```

これらは、ツールバー、ロケーションバー、ステータスバー、メニューバー、スクロールバーの表示をそれぞれ消すことを意味しています。もちろん、消したいバーについてだけ設定することも可能です。また、ウィンドウのサイズもwidth、heightを指定することにより行っています。数値はピクセルです。

ウィンドウを開くにあたって指定できる主な項目には、以下のようなものがあります。それぞれの項目はカンマで区切って並べてください。

オプション	書　式	初期値	機　能
channelmode	channelmode[=yes|no][=0|1]	no	チャンネルモード
directories	directories [=yes|no][=0|1]	yes	ディレクトリバーの表示/非表示
fullscreen	fullscreen [=yes|no][=0|1]	no	フルスクリーンモード
height	height = <pixel>	–	ウインドウの高さ
left	left = <pixel>	–	デスクトップ左端からの座標
location	location [=yes|no][=0|1]	yes	ロケーションバーの表示/非表示
munebar	menubar [=yes|no][=0|1]	yes	メニューバーの表示/非表示
resizable	resizable [=yes|no][=0|1]	yes	ウィンドウサイズの変更可/不可
scrollbars	scrollbars [=yes|no][=0|1]	yes	スクロールバーの表示/非表示
status	status [=yes|no][=0|1]	yes	ステータスバーの表示/非表示
titlebar	titlebar [=yes|no][=0|1]	yes	タイトルバーの表示/非表示
toolbar	toolbar [=yes|no][=0|1]	yes	ツールバーの表示/非表示
top	top = <pixel>	–	デスクトップ上端からの座標
width	width = <pixel>	–	ウインドウの幅

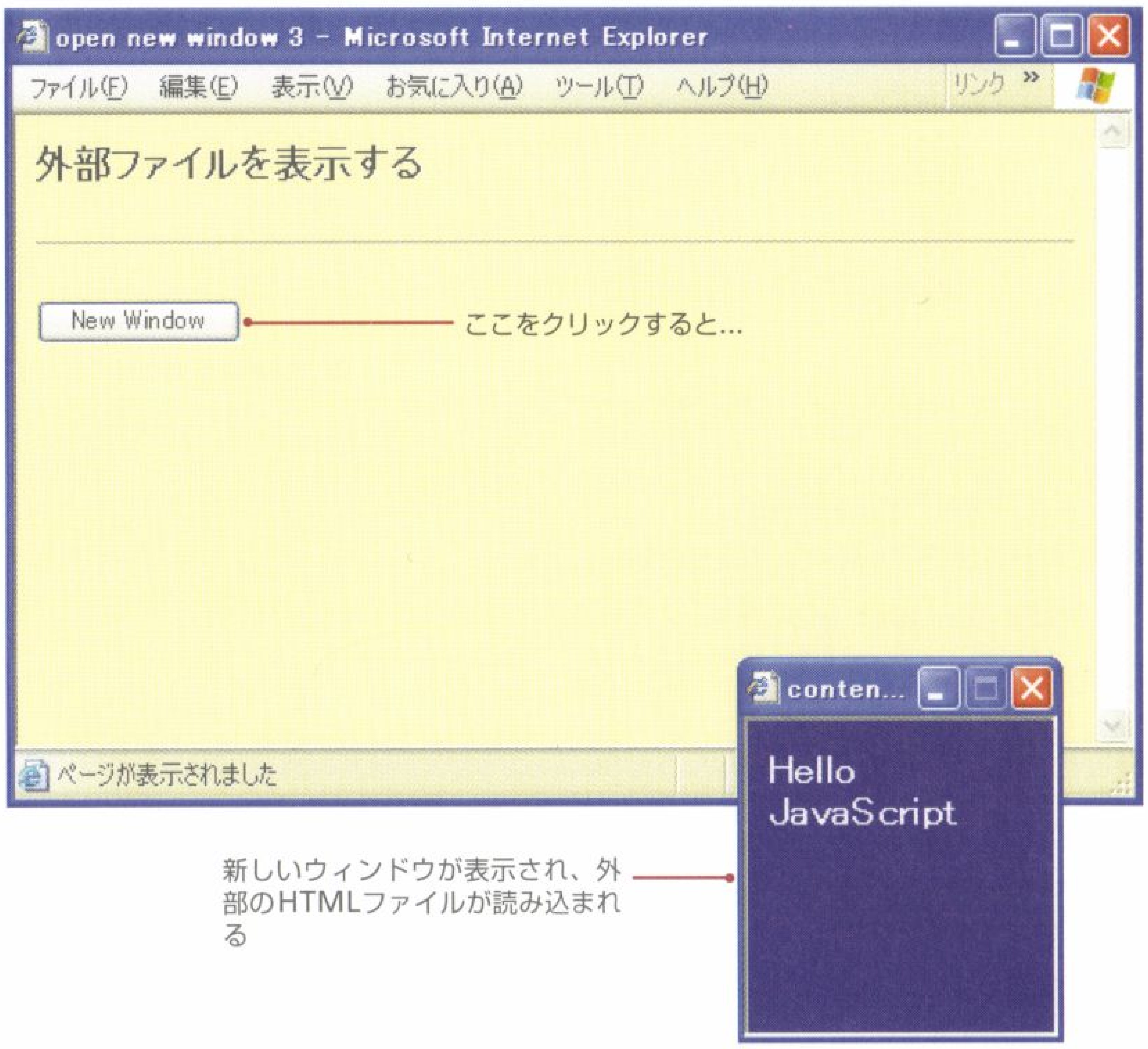
また②の部分では、新しく開くウィンドウのバックグラウンドカラーを指定するとともに、「Hello New Window」の文字がウィンドウの中央に表示されるようなHTMLソースを複数のwriteメソッドにより書き込んでいます。本来、このような処理の場合は、次項で紹介する外部ファイルを読み込む方法が一般的ですが、何らかの理由で外部ファイルを使いたくない場合は、このようにwriteメソッドですべてのHTMLソースを書き込む方法もあります。

ウィンドウを開き、外部ファイルを読み込む

```
window.open('URL','ウィンドウ名','スタイル');
```

新しくウィンドウを生成するときに、すでにあるHTMLファイルを読み込んで表示するスクリプトです。通常ウィンドウを開く場合にはもっとも多い使い方でしょう。

DISPLAY



SOURCE

newwindow3.html

```
<html>
<title>open new window 3</title>
<script language="javascript">
<!--
function openWin(){
```

```
    var win1 = window.open('openwincontents.html',
⇒       'newwin1','toolbar=no,location=no,status=no,
⇒       menubar=no,scrollbars=no,width=150,height=150');   ①
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>外部ファイルを表示する</h3>
<hr />
<form>
    <input type="button" value="New Window" onclick="openWin()"
/>
</form>
</body>
</html>
```

openwincontents.html

```
<html>
<title>contents</title>
<body bgcolor="navy" text="white">
<h3>Hello JavaScript</h3>
</body>
</html>
```

DETAILS

ここでもフォームボタンをクリックすることにより、openWin関数が呼び出されて新しいウィンドウが表示されます。ただし、これまでに紹介したウィンドウを開くスクリプトでは、openメソッドで新しいウィンドウを生成した後、writeメソッドで表示する内容を書き込んでいましたが、このスクリプトでは別のHTMLファイルを新たなウィンドウに読み込んでいます。

openWin関数を見てみましょう。openWin関数ではwindowオブジェクトのopenメソッドによりウィンドウを生成しています。記述は次のようになっています(①の箇所)。

```
var win1 = window.open('openwincontents.html', 'newwin1',
➡  'toolbar=no,location=no,status=no, menubar=no,
➡  scrollbars=no,width=150,height=150');
```

これまで紹介した例と異なっているのは、1つ目のシングルクォーテーション内に、読み込むファイルのURLが記述してあることです。ここでは、「openwincontents.html」というファイル名を指定しています。次に、ウィンドウに対して「newwin1」という名前を付けています。

最後に、ウィンドウのさまざまな表示設定を行っています。ここでは、ウィンドウに表示される一連のバーを消すのに加えて、新しく開くウィンドウのサイズが幅150ピクセル、高さが150ピクセルになるように指定を行っています。詳しくはP.128を参照してください。

T I P S

インターネットの業界では、最近「ポップアンダー広告」というものがあります。いくつかのサイトでは、ユーザーがサイトを訪れたときに新しいウィンドウを開いてポップアップ広告を出します。これに対してポップアンダー広告では、ユーザーが見ているWebブラウザの裏側にもう1つウィンドウを開いて、そこにコンテンツを表示するということを行います。実現方法としては、開いたウィンドウに対して、blurメソッドを用いてウィンドウからフォーカスを外します。実際には以下のような記述になります。

```
var win1 = window.open('openwincontents.html', 'newwin1',
➡  'toolbar=no,location=no,status=no, menubar=no,
➡  scrollbars=no,width=150,height=150');
win1.blur();
```

しかし、この手法はあまり用いないほうがよいでしょう。確かにユーザーの目にはとまる手法ですが、反感を買う可能性もあります。

4-4 WIN IE 4↑ N 6↑ MAC IE 4↑ N 6↑

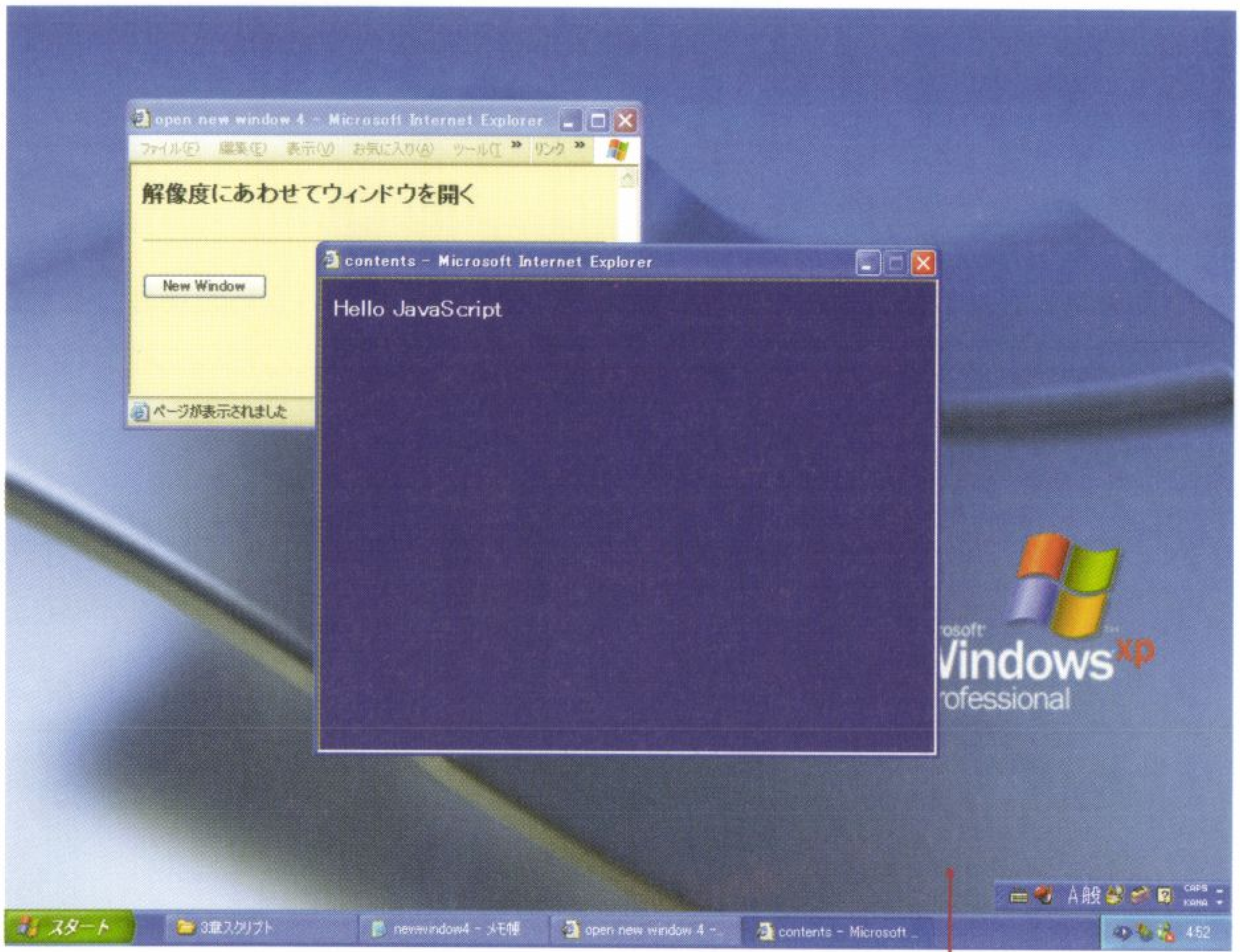
ウィンドウを解像度に合わせて開く

screen.width

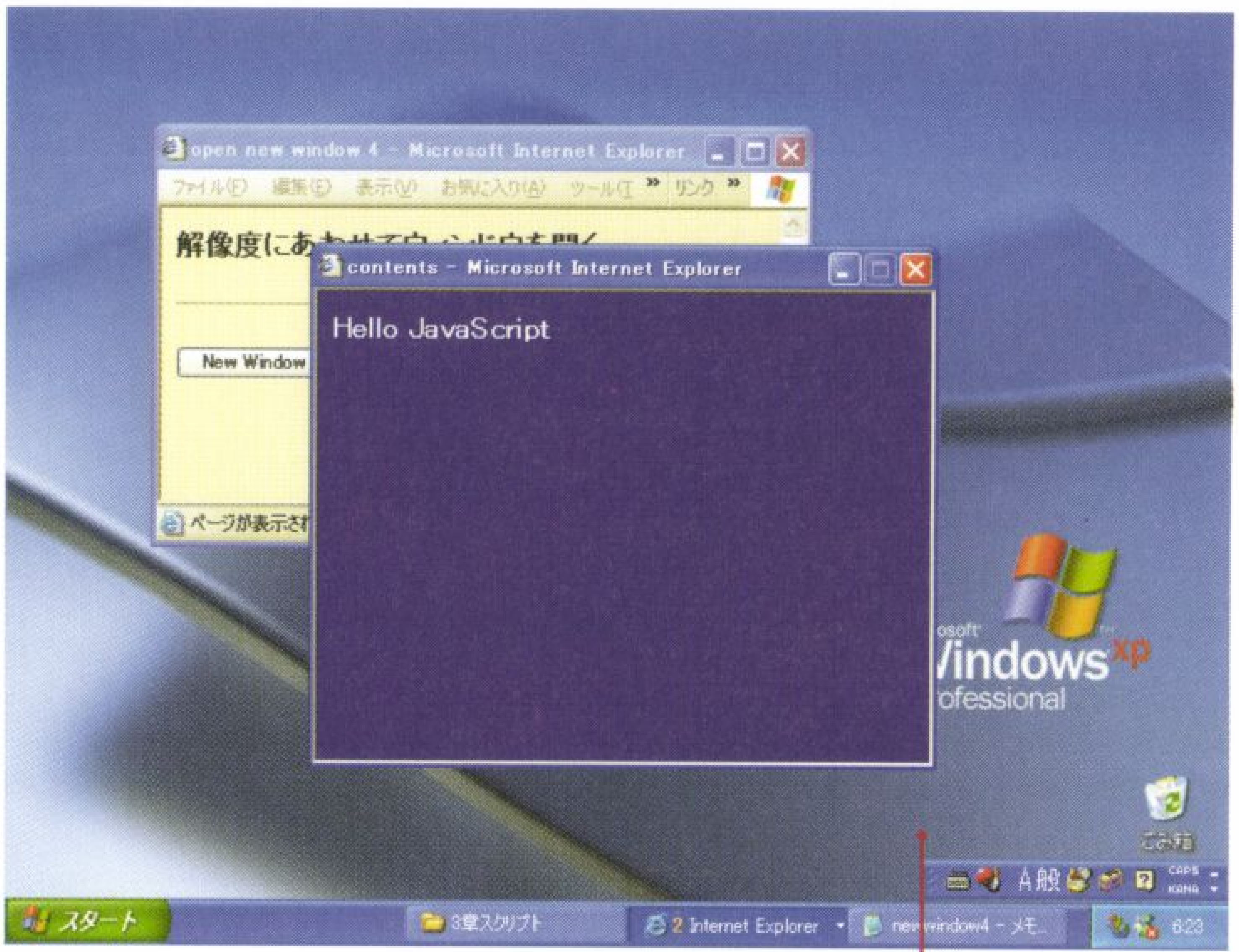
screen.height

ウィンドウを開くのに、サイズ、位置を指定できるのはすでに解説したとおりです。ここではさらに発展させて、Webブラウザが表示されている画面のサイズ（解像度）に合わせてウィンドウを表示させてみます。

DISPLAY



1024×768ピクセルの場合



800×600ピクセルの場合

SOURCE newwindow4.html

```
<html>
<title>open new window 4</title>
<script language="javascript">
<!--
var displayWidth = screen.width;            ─①
var displayHeight = screen.height;

function openWin(){
    var winWidth = displayWidth/2;          ─②
    var winHeight = displayHeight/2;
    var winLeft = displayWidth/4;
    var winTop = displayHeight/4;
    var optionString = "toolbar=no,location=no,status=no,
⇒       menubar=no,scrollbars=no,";
    optionString += "width=" +winWidth + ", height=" +        ─③
⇒       winHeight + ",left=" + winLeft + ",top=" + winTop ;
    var win1 = window.open('openwincontents.html', 'newwin1',  ─④
⇒       optionString);
}
```

```
//-->
</script>
<body bgcolor="#fff8dc">
<h3>解像度にあわせてウィンドウを開く</h3>
<hr />
<form>
   <input type="button" value="New Window" onclick="openWin()"
⇒       />
</form>
</body>
</html>
```


DETAILS

このサンプルでは、ディスプレイの解像度に合わせて生成するウィンドウをコントロールしています。具体的にはディスプレイの解像度を調べた後、ディスプレイの解像度の幅、高さのそれぞれ半分の長さを持つウィンドウを、ディスプレイの中心に開きます。そのためには、ウィンドウの左上端を、画面の左隅から横に1/4、縦に1/4ずらした位置に開く必要があります。



まずディスプレイの解像度を調べるには、screenオブジェクトのwidthプロパティとheightプロパティを用います。サンプルでは、それぞれのプロパティから取得した値を、変数displayWidth、変数displayHeightに格納しています(①の箇所)。

そして、それらを基にして画面の幅の2分の1、4分の1、画面の高さの2文の1、4分の1の値を計算し、変数winWidth、変数winHeight、変数winLeft、変数winTopに格納しています。たとえば、画面解像度が800×600ピクセルの場合は、次のような値が各変数に格納されます(②の箇所)。

```
displayWidth=800
displayHeight=600
winWidth=400
winHeight=300
winLeft=200
winTop=150
```

これらの変数の値を、ウィンドウを開くときのスタイルを指定を格納している変数optionStringの文字列に連結し(③の箇所)、その指定を用いてウィンドウを開きます(④の箇所)。たとえば、画面解像度が800×600ピクセルのとき、実行されるスクリプトは、次のようになります。

```
var win1 = window.open('openwincontents.html', 'newwin1',
➡ toolbar=no,location=no,status=no,menubar=no,scrollbars=
➡ no,width=400,height=300,left=200,top=150);
```

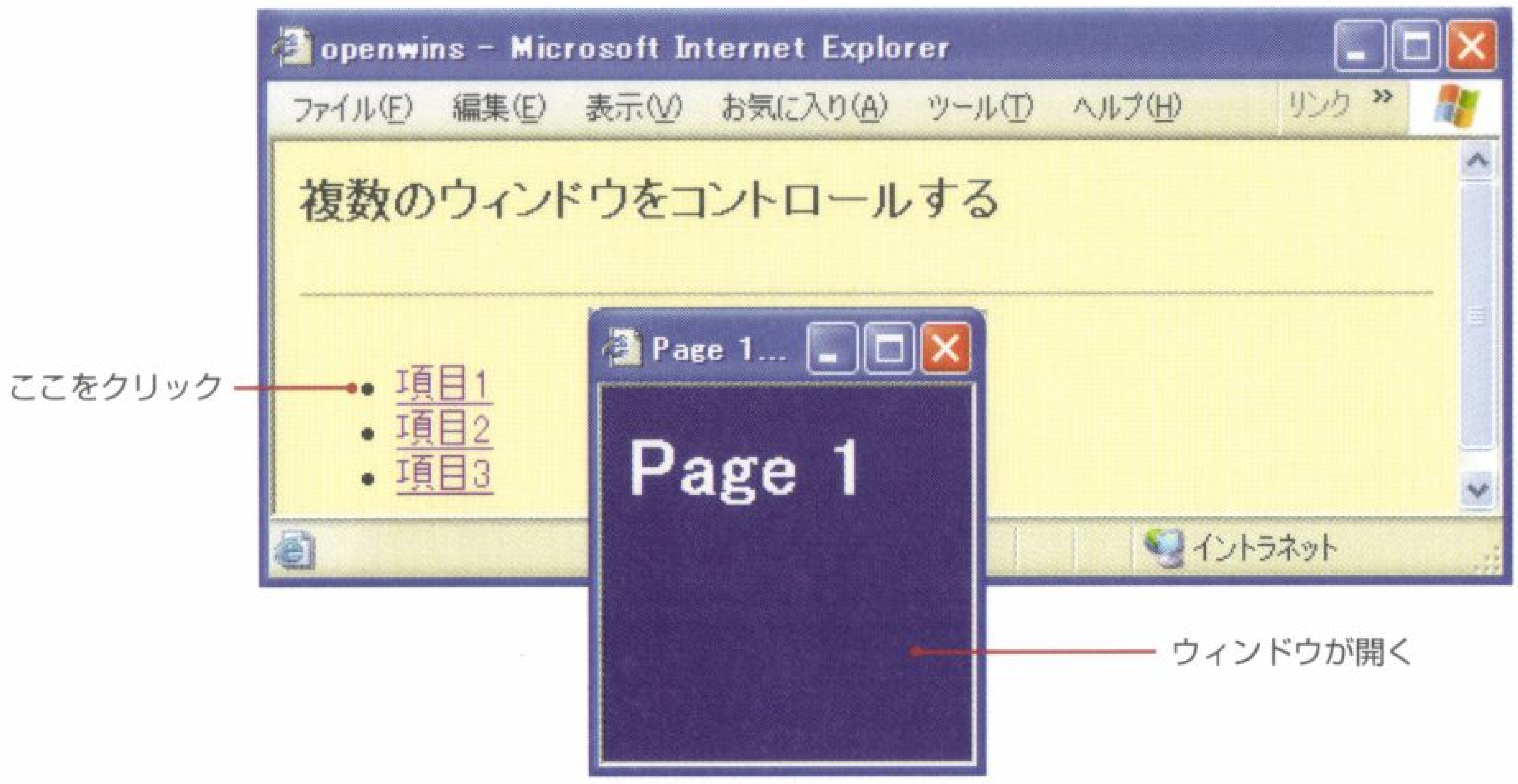
このようにすると、ディスプレイの解像度に合わせてウィンドウを開くことができます。

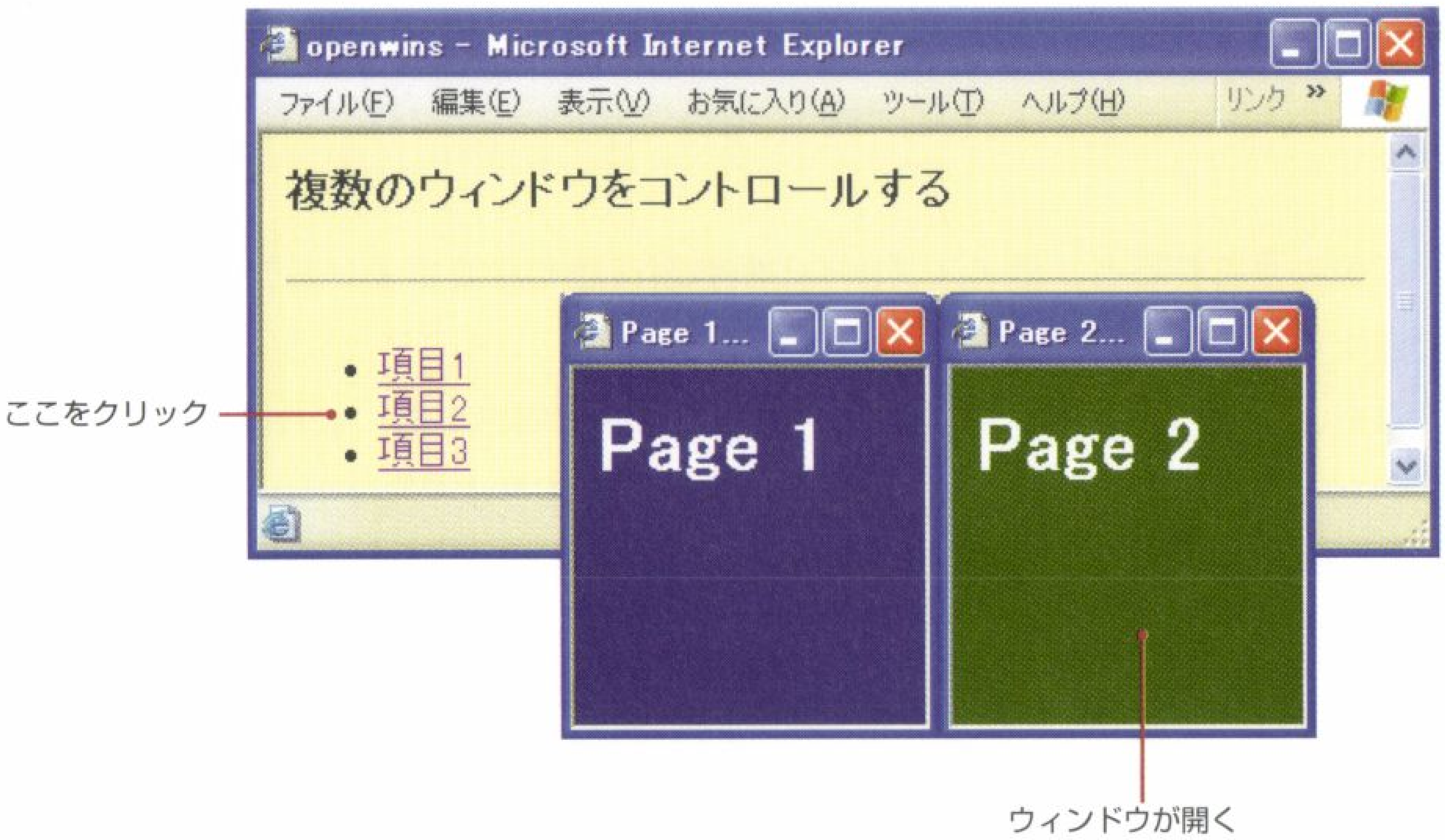
なお、IEでウィンドウをフルスクリーンで開く場合には、screenオブジェクトのwidthプロパティ、heightプロパティを使うよりも、openメソッドのfullscreenオプションを用いるのがよいでしょう。

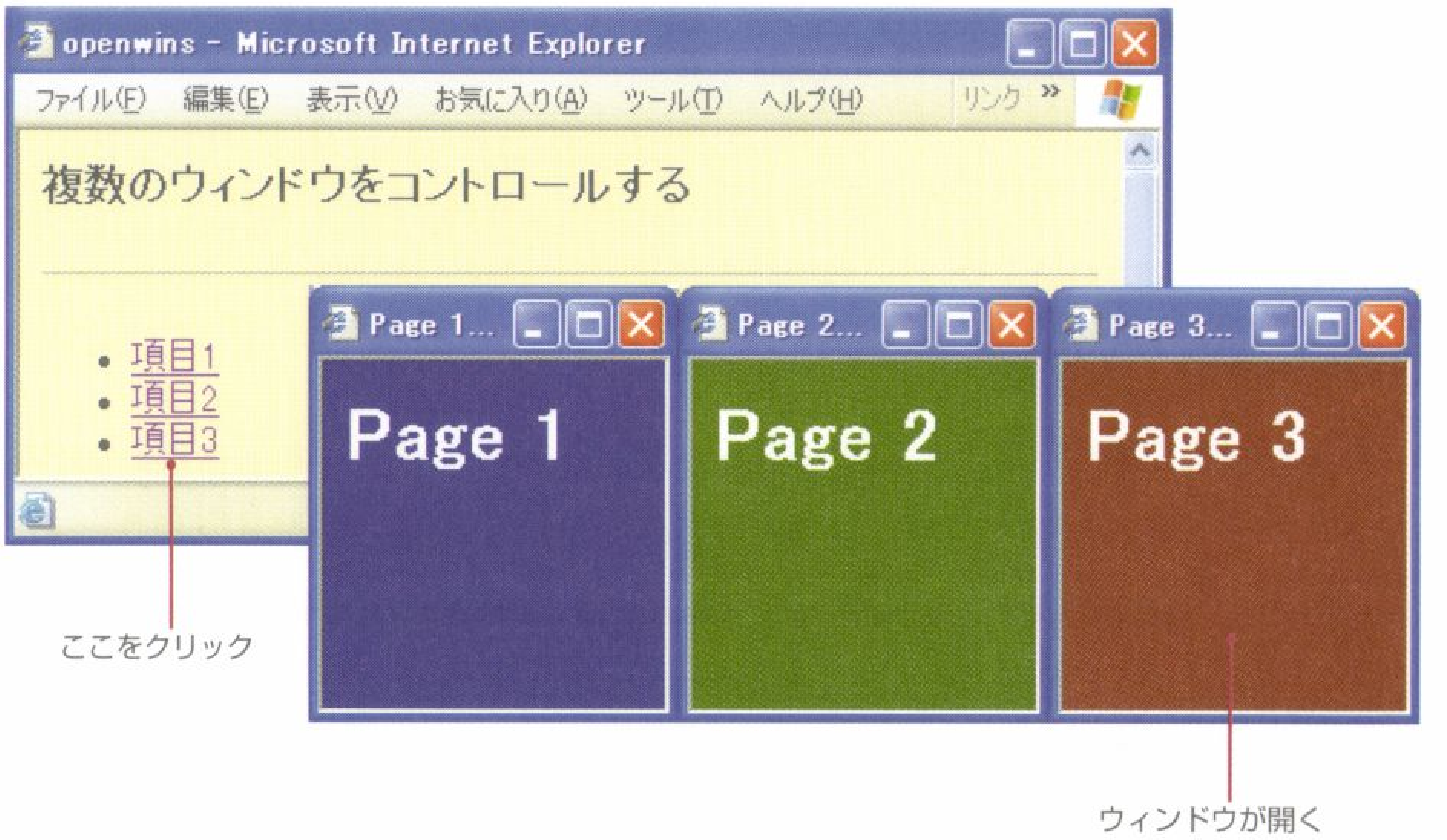
複数のウィンドウを開く

ここまでウィンドウの開き方を解説してきましたが、どれも1つのウィンドウに対してのコントロールでした。しかし、1つだけしかウィンドウを開けないわけではありません。複数のウィンドウを開くサンプルを紹介します。

DISPLAY







SOURCE

windowcontrol3.html

```
<html>
<title>openwins</title>
<script language="javascript">
<!--
function openWindow(clocation, wNo){ ―①
   var wName;
   wName = "win" + wNo; ―②
   var scriptwin = window.open(clocation,wName,'toolbar=no,  ┐
➡      location=no,resizable=yes,status=no,width=150,height=  ├③
➡      150');
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>複数のウィンドウをコントロールする</h3>
<hr />
<ul>
   <li><a href="javascript: openWindow('test1.html','1');">  ┐
➡      項目1</a></li>                                          ┘④
   <li><a href="javascript: openWindow('test2.html','2');">
➡      項目2</a></li>
```

```
    <li><a href="javascript: openWindow('test3.html','3');">
➡       項目3</a></li>
</ul>
</body>
</html>
```

D E T A I L S

このサンプルでは「項目1」、「項目2」、「項目3」というリンクを張られた文字列があり、それぞれのリンクをクリックすると、それぞれ新しいウィンドウが表示されます。ウィンドウを開くには、openWindow関数を呼び出します。

このサンプルで使用するopenWindow関数は、これまで登場したものと違って、clocationとwNoという引数を2つ持っています。clocationは、開いたウィンドウに読み込むWebページのURLで、wNoはウィンドウを識別するための番号です（①の箇所）。

openWindow関数内では、引数で与えられたウィンドウを識別するための番号を基にウィンドウの名前を生成しています。ここでは「win」という文字列にウィンドウ番号（wNo）を連結し、「wName」に代入しています（②の箇所）。

その後、変数clocationに格納されたURLと生成された名前wNameを基に、windowオブジェクトのopenメソッドを用いてウィンドウを開いています（③の箇所）。たとえば④の部分において、実際に実行されるopenWindow関数の中身は、次のようになっています。

```
var scriptwin = window.open('test1.html','win1',
➡  'toolbar=no,location=no,resizable=yes,status=no,
➡  width=150,height=150');
```

このように、生成するウィンドウ名が異なると、すでにウィンドウが開いていても新たにウィンドウを生成することができます。

用語解説などで複数のウィンドウを開く必要がある場合に、上手に使ってください。ただし、あまりたくさんウィンドウを開きすぎると、邪魔になるだけでなく動作が重くなってしまうといったことも起こる可能性があります。

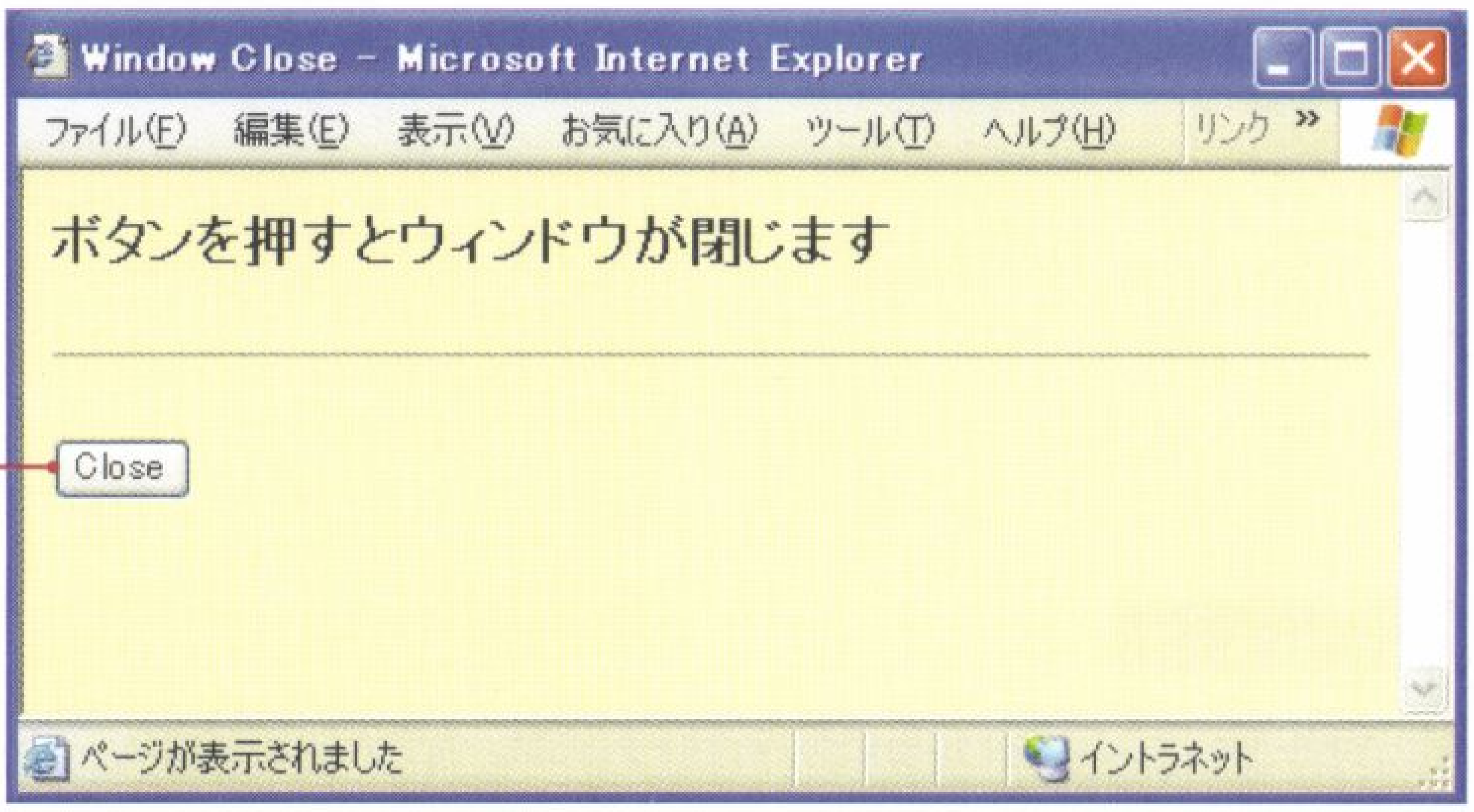
4-6 WIN MAC

ボタンクリックでウィンドウを閉じる

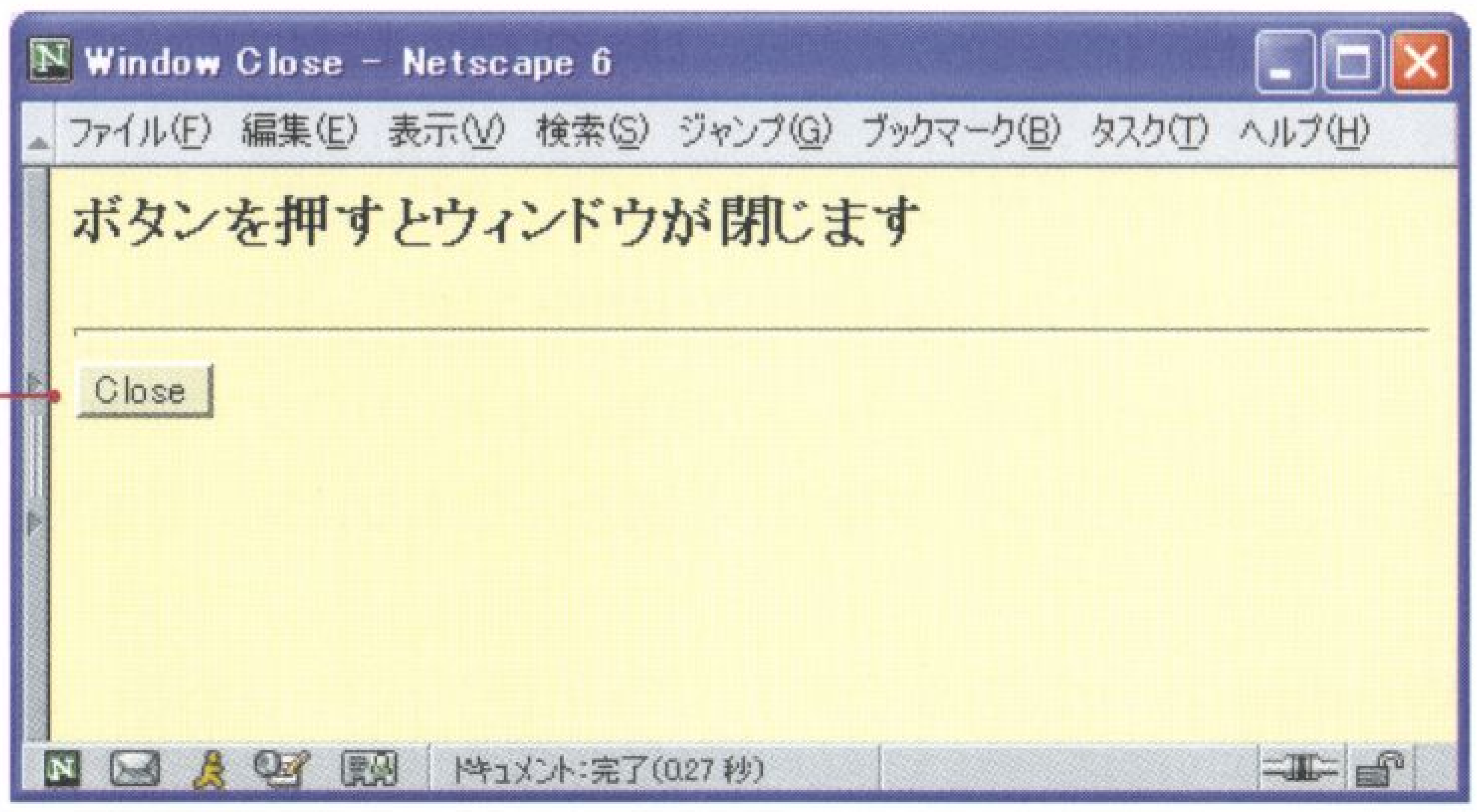
window.close()

JavaScriptを使うと、自分自身のウィンドウを閉じて終了させることが可能です。このサンプルは、自分自身を閉じるというスクリプトです。

DISPLAY



ここをクリックすると
ウィンドウが閉じる



SOURCE windowclose.html

```
<html>
<title>Window Close</title>
<body bgcolor="#fff8dc">
<h3>ボタンを押すとウィンドウが閉じます</h3>
<hr />
<form>
   <input type="button" value="Close"
      onclick= "window.close()" />
</form>
</body>
</html>
```

DETAILS

Webページには、「close」と表示されたフォームボタンが配置されています。このボタンをクリックすると、Webページが閉じられます。

ボタンには、<input>タグのonclickイベントハンドラによって、ウィンドウを閉じるためのメソッドが呼び出されています。ウィンドウを閉じるには以下のように記述します。

```
window.close()
```

これは、windowオブジェクトに対して使用するメソッドです。このcloseメソッドが行われたwindowオブジェクトは閉じます。パラメータの指定はありません。新しく生成したウィンドウや、別のウィンドウに対しても実行できます。

なお、Windows版のIE4以降では、デフォルトウィンドウを閉じようとすると、確認のダイアログが表示されます。これはセキュリティ対策のために導入されました。



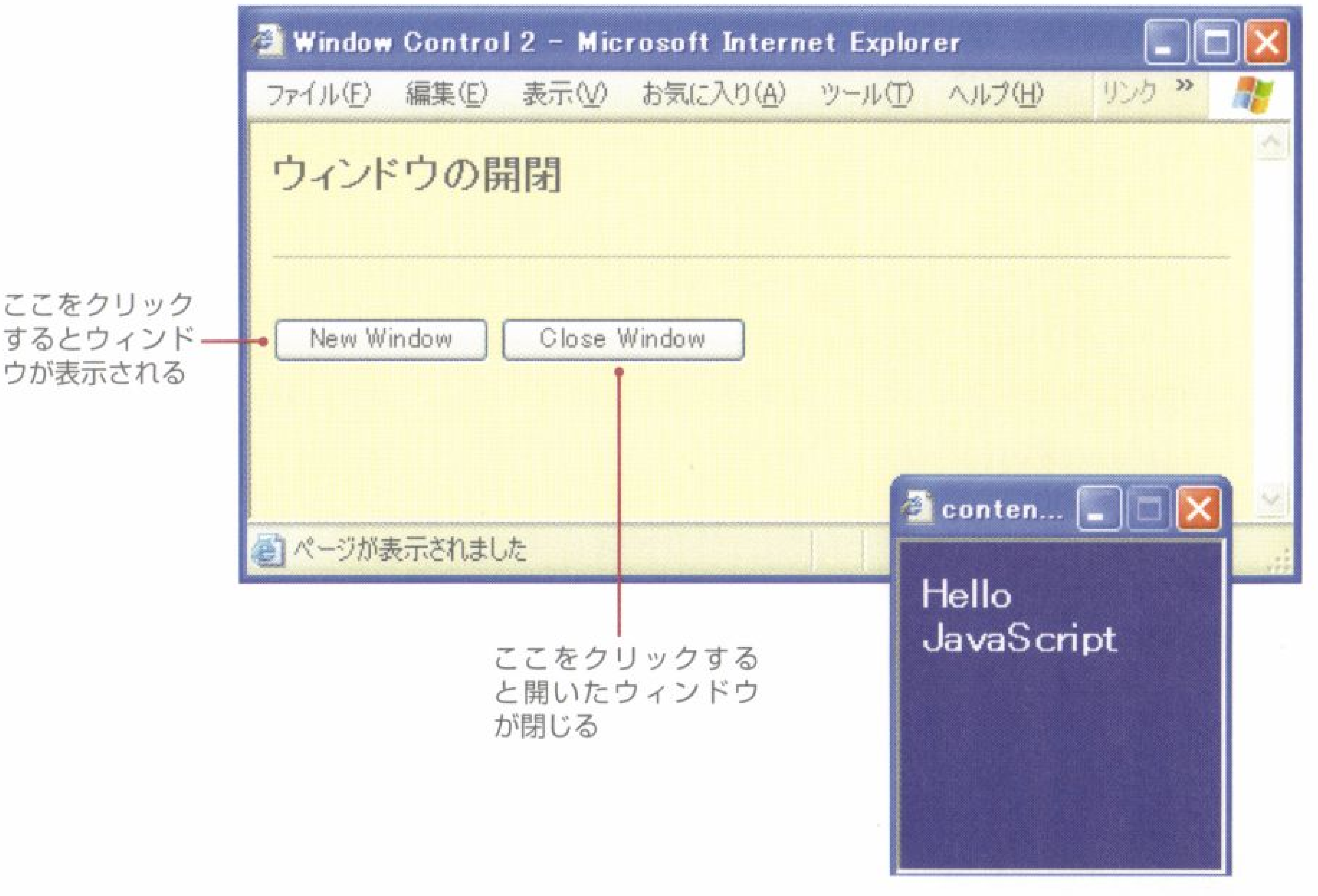
確認のためのダイアログボックス

ボタンクリックでウィンドウを開閉する

`windowオブジェクト名.close();`

新たなウィンドウを開くだけではなく、ウィンドウを閉じることも大事な作業の1つです。JavaScriptでは、当然のことながらどちらの作業も行うことができます。ここでは、それぞれの作業を行うスクリプトを紹介します。

DISPLAY



SOURCE windowcontrol.html

```
<html>
<title>Window Control 2</title>
<script language="javascript">
<!--
var win1;

function openWin(){
   win1=window.open('openwincontents.html','newwin',
      'toolbar=no,location=no,status=no,menubar=no,
      scrollbars=no,width=150,height=150');
}
```

```
function closeWin(){
   if(win1.closed == false){
      win1.close();
   }                                                          ②
   else{
      alert("すでにウィンドウは閉じられています");
   }
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>ウィンドウの開閉</h3>
<hr />
<form>
   <input type="button" value="New Window" onclick=
➡     "openWin()" />
   <input type="button" value="Close Window" onclick=         ①
➡     "closeWin()" />
</form>
</body>
</html>
```

DETAILS

このスクリプトは、これまで紹介したウィンドウを開くスクリプトと、閉じるスクリプトを組み合わせたものです。Webページ上には2つのフォームボタンが配置されており、左側の[New Window]はウィンドウを開くボタン、右側の[Close Window]はウィンドウを閉じるボタンです。

[New Window]をクリックすると、<input>タグのonclickイベントハンドラによってopenWin関数が呼び出されます。openWin関数はwindowオブジェクトのopenメソッドを用いて新たにウィンドウを開きます。

また、このスクリプトでは親ウィンドウから子ウィンドウを消すという機能も実装しています。ウィンドウを消す操作は[Close Window]をクリックしてcloseWin関数を呼び出すことで行います(①の箇所)。

closeWin関数ではまず、ウィンドウを消す対象となるウィンドウが開かれているかどうかをif文により確認します。この確認にはwindowオブジェクトのclosedプロパティを用います。closedプロパティは対象となるウィンドウが閉

じられている場合にはtrueを、開かれている場合にはfalseを返します。そのため、closedプロパティの値がfalseの場合にウィンドウを閉じる処理を行います(②の箇所)。ウィンドウを閉じるにはwindowオブジェクトのcloseメソッドを用います。

```
windowオブジェクト名.close()
```

P.139で紹介したものはwindowオブジェクトとして、呼び出したウィンドウ自身を表す「window」を使っていましたが、このスクリプトでは生成したウィンドウのオブジェクトである「win1」を用います。そのため、ウィンドウを閉じる部分の記述は以下のようになります。

```
win1.close()
```

win1というwindowオブジェクトに対してcloseメソッドを使用することによって、openWin関数により生成されたウィンドウが閉じられます。

またウィンドウがすでに閉じられている場合は、「else {」以下の命令が実行されてアラートが表示されます。



今回のサンプルでは動作の確認のために、アラートを表示していますが、実際に用いる際には、アラートを表示する必要はないでしょう。また、同様にウィンドウを閉じる処理を行うときには、必ずclosedプロパティによりウィンドウの状態を確認するようにしましょう。

4-8 WIN MAC

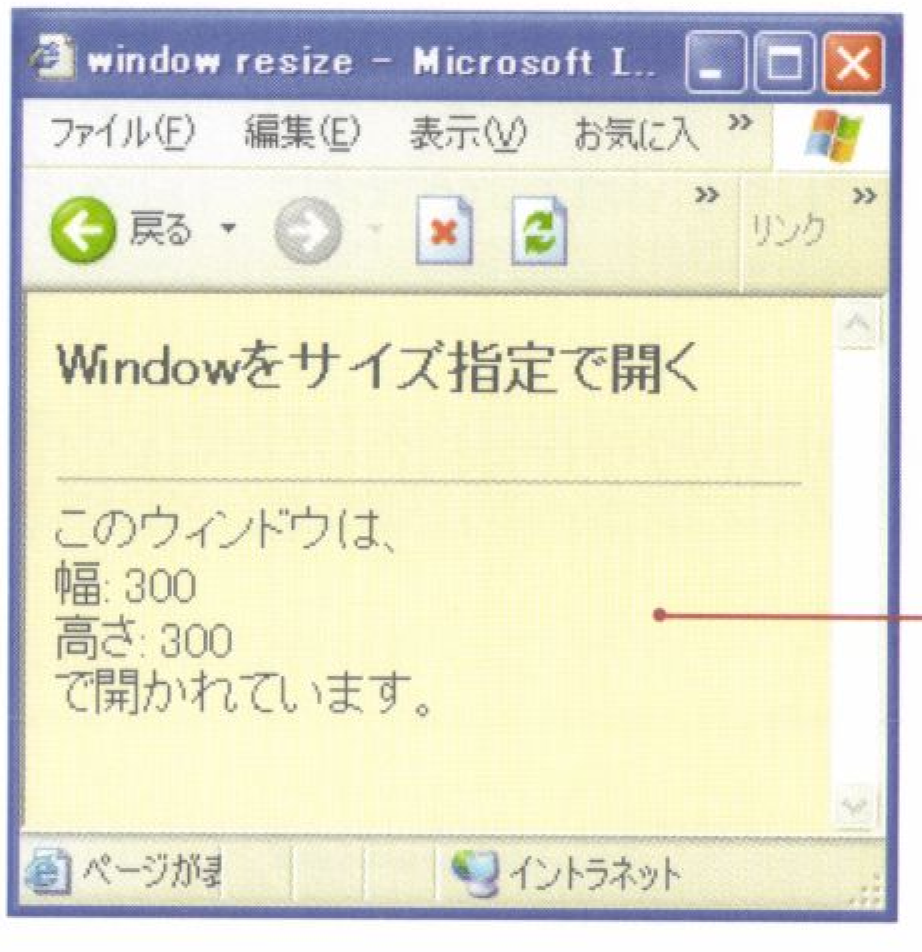
ウィンドウのサイズを変更する

windowオブジェクト名.resizeTo(ウィンドウの横幅,ウィンドウの縦幅)
windowオブジェクト名.outerWidth=ウィンドウの横幅
windowオブジェクト名.outerHeight=ウィンドウの縦幅

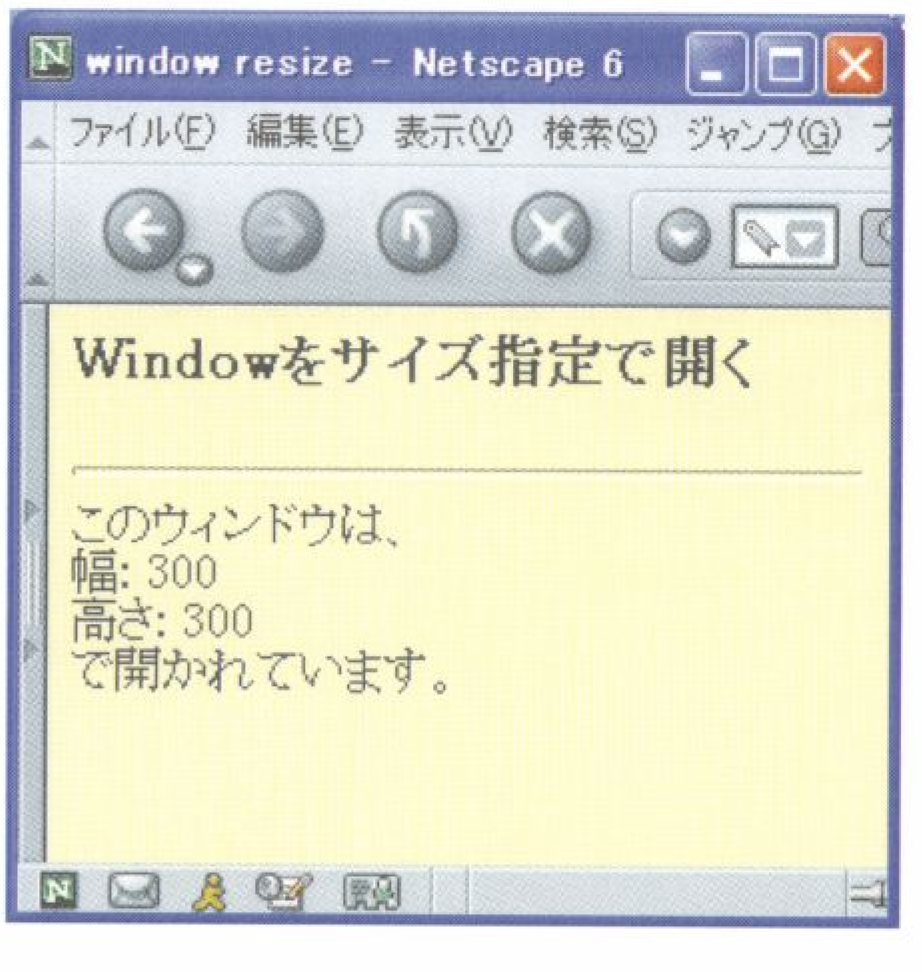
Webページロード時に、ウィンドウのサイズを、指定したサイズに変更するスクリプトです。新たにウィンドウを開くときには、openメソッドを用いてサイズを指定することができますが、すでに開いているウィンドウのサイズを変更することはできません。しかしこのスクリプトでは、Webページをロードするとともに、すでに開いているウィンドウ自体のサイズを変更することができます。

WINDOW

DISPLAY



ウィンドウが開いた後、
自動的にリサイズされる



SOURCE resizewindow1.html

```
<html>
<title>window resize</title>

<script language="javascript">
<!--
var ua = navigator.userAgent;
var winWidth = 300; //Windowの幅
var winHeight= 300; //Windowの高さ

if( ua.indexOf("MSIE",0) >=0 ){                                ②
   window.resizeTo(winWidth, winHeight);
}
else if( ua.indexOf("Netscape6",0)>=0 || ua.indexOf(
   "Mozilla/4",0)>=0){                                         ③
   window.outerWidth = winWidth;                 ①
   window.outerHeight = winHeight;
}
//-->
</script>

<body bgcolor="#fff8dc">
<h3>Windowをサイズ指定で開く</h3>
<hr />
このウィンドウは、<br />

<script language="javascript">
<!--
   document.write("幅: " + winWidth + "<br />");
   document.write("高さ: " + winHeight + "<br />");
//-->
</script>

で開かれています。
</body>
</html>
```

DETAILS

このサンプルでは、Webページがロードされるとウィンドウが縦300ピクセル、横300ピクセルにリサイズされます。しかし、ただ単純にリサイズするメソッドを実行しているだけではありません。

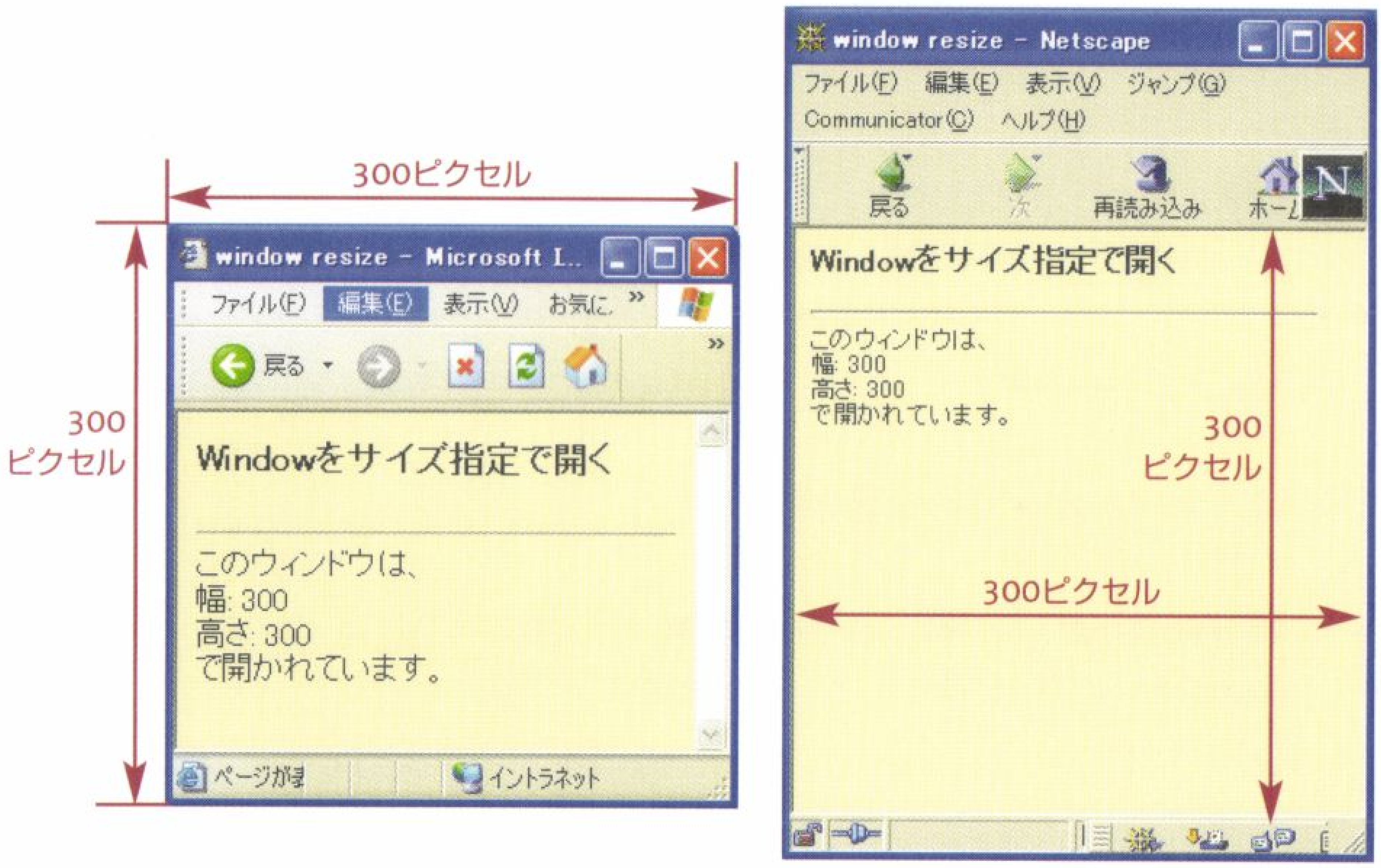
ウィンドウをリサイズする場合は、windowオブジェクトのresizeToメソッドを、次の書式で使用します。

```
windowオブジェクト名.resizeTo(ウィンドウの横幅,ウィンドウの縦幅)
```

しかしながらresizeToメソッドはちょっとした問題を抱えていて、IEとバージョン4以前のNNではリサイズされたあとのサイズが異なります。たとえば、以下のような記述でリサイズしたとします。

```
window.resizeTo(300, 300);
```

このコードを含むHTMLファイルをIE6とNN4で表示させると、下の図のようになってしまいます。



これは、resizeToメソッドのサイズの解釈がそれぞれ異なるからです。IEではウィンドウの外枠、NN4以前のバージョンではドキュメント部分の大きさと解釈されます。NN6ではIEと同じように解釈されますので問題ありませんが、NN4以前のバージョンを使用しているユーザーを切り捨てるわけにはいきません

ので、resizeToメソッドだけを単純に使用することはできません。

そこで、NNに用意されているouterHeightプロパティとouterWidthプロパティを使用します(①の箇所)。

```
windowオブジェクト名.outerWidth=ウィンドウの横幅
windowオブジェクト名.outerHeight=ウィンドウの縦幅
```

サンプルではこれらを用いて、どちらのWebブラウザでも同じように処理されるスクリプトにしています。

本サンプルでは、IEとNNの処理を分けて記述しています。その分離に用いているのが②と③の部分です。IEまたはMSN ExplorerはuserAgentプロパティ(P.111参照)の文字列に「MSIE」が必ず入っているため、まずこの文字列で取り出します。取り出したIEではresizeToメソッドが実行されます。その後でNNは、NN4が「Mozilla/4」、NN6が「Netscape6」という文字列が入っているので、これで取り出します。その結果、outerHeightプロパティとouterWidthプロパティにそれぞれ高さと幅が設定されます。

Webブラウザの判定は、必ずIEまたはMSN Explorerの判定から行ってください。もしも文字列「Mozilla/4」での判定を最初に行ってしまうと、IEやMSN Explorerも取り出されてしまいます。これはIEやMSN Explorerにも文字列「Mozilla/4」が含まれているためです。Webブラウザの判定方法に関する詳細は、P.108を参照してください。

このようにしてWebブラウザごとに記述を分けることにより、IEとNNで同様に動作するリサイズのスクリプトを実現しています。

なお、IEに実装されているJScriptにはresizeToメソッドは存在しません。実はIEでは、resizeToメソッドはDHTMLに実装されています。そのため、リファレンスなどを探す場合には気を付けてください。

4-9

WIN IE 4↑ N 4↑ MAC IE 4↑ N 4↑

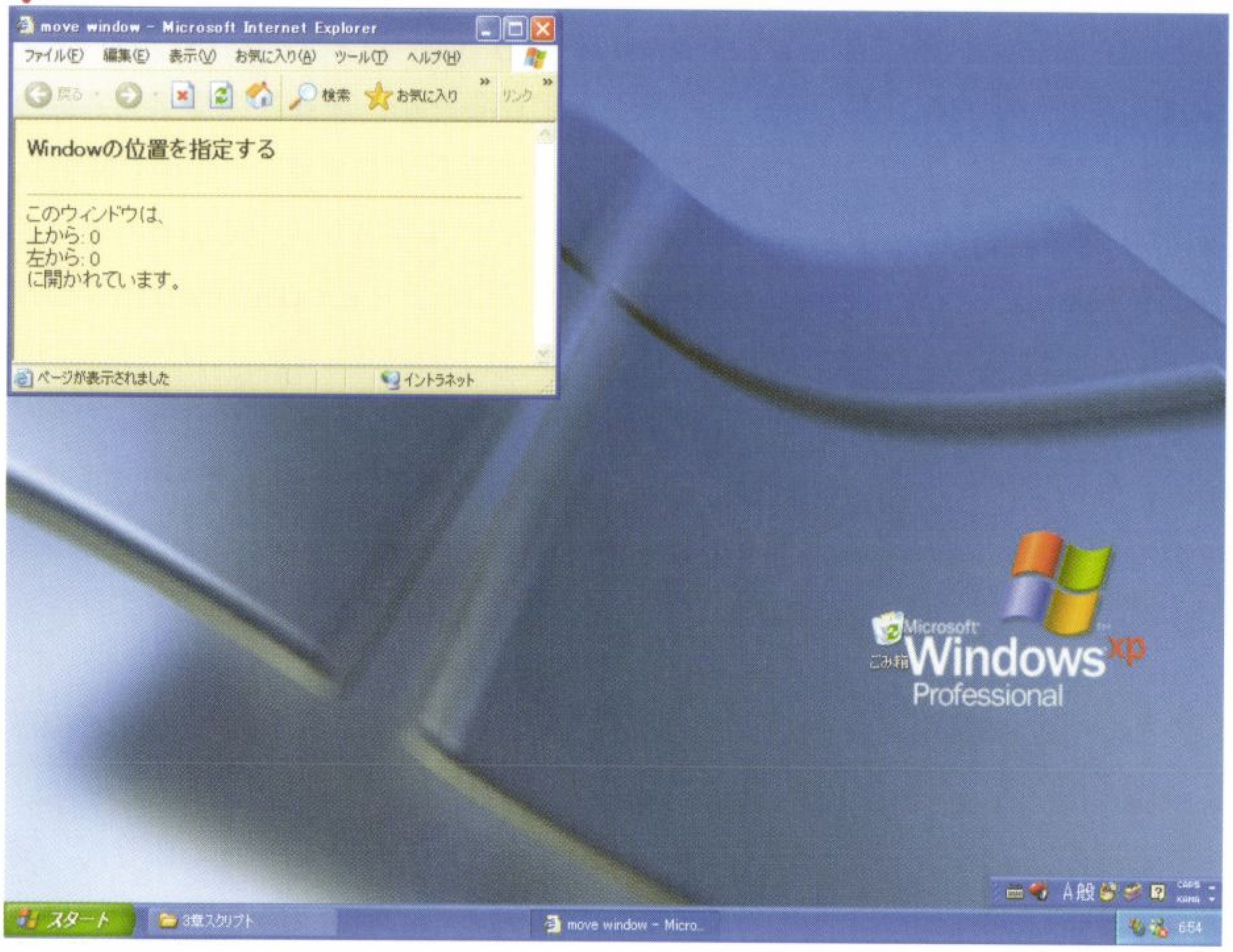
ウィンドウの位置を変更する

windowオブジェクト名.moveTo(x座標,y座標);

ウィンドウを、デスクトップのどこに表示させるか指定することができるスクリプトです。複数のウィンドウを配置する場合などに使うと便利でしょう。

DISPLAY

必ずこの位置に表示される



SOURCE

movewindow1.html

```
<html>
<title>move window</title>

<script language="javascript">
<!--
var winTop = 0;
var winLeft = 0;            ①
window.moveTo(winTop, winLeft);
//-->
</script>

<body bgcolor="#fff8dc">
<h3>Windowの位置を指定する</h3>
<hr />
このウィンドウは、<br />

<script language="javascript">
   document.write("上から: " + winTop + "<br />");
   document.write("左から: " + winLeft + "<br />");
//-->
</script>

に開かれています。
</body>
</html>
```

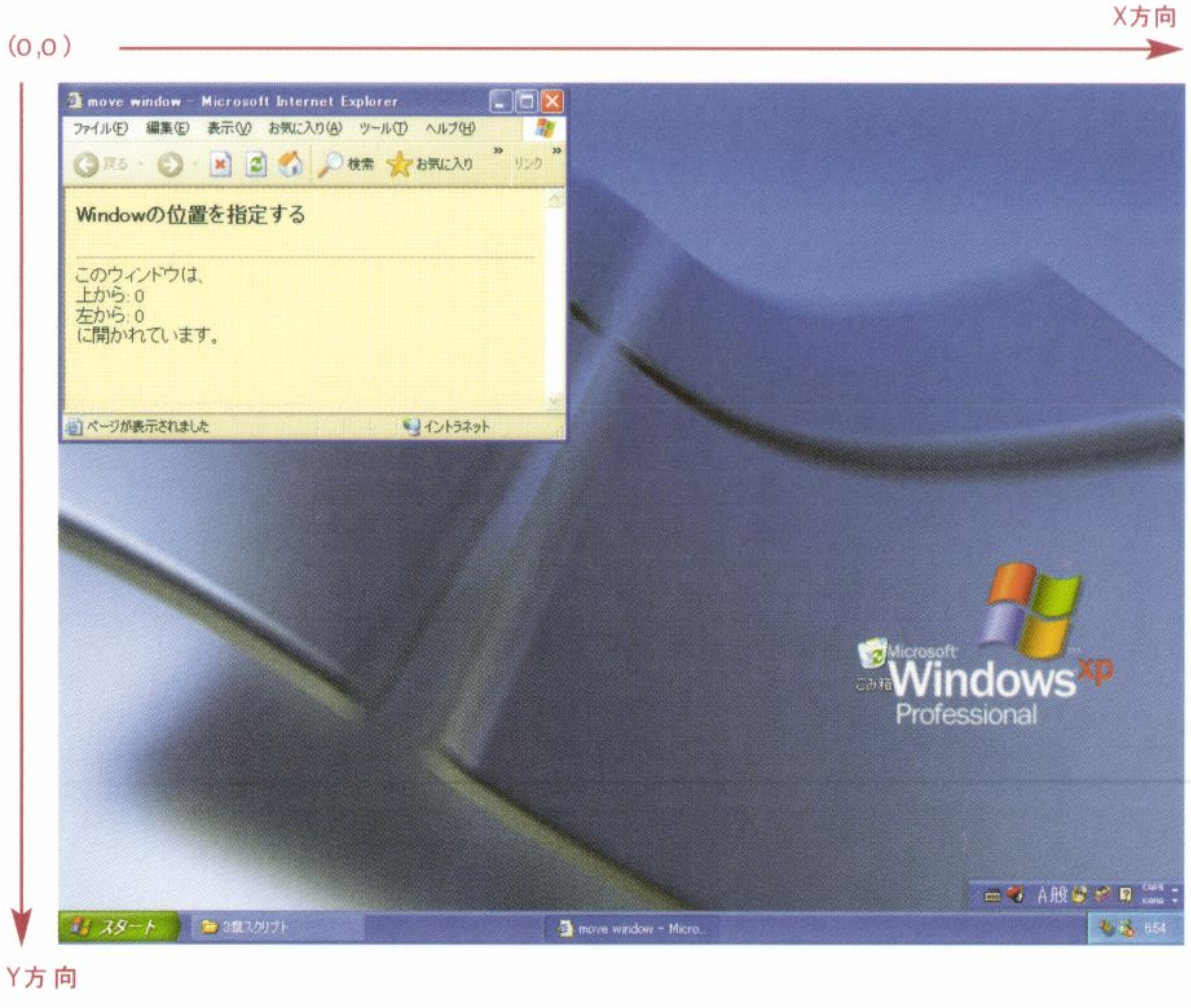
DETAILS

このサンプルでは、Webページのロード時にウィンドウの位置を左隅に移動しています。ウィンドウを動かすメソッドはmoveToメソッドです。moveToメソッドの書式は、以下のようになります。

```
windowオブジェクト名.moveTo(x,y);
```

ここでx、yは、位置を指定したいウィンドウの左上端の座標を指定します。座標は画面の左上が(0,0)で右方向がX方向、下方向がY方向です。

このサンプルでは、①の部分で変数winTop、winLeftともに値を「0」にしているので、指定は(0,0)と同じになります。そのため、画面左上にウィンドウが表示されます。



moveToメソッドは、IEに実装されているJScriptには存在しません。しかしなぜ動作するかというと、IE版DHTMLにmoveToメソッドが実装されているためです。

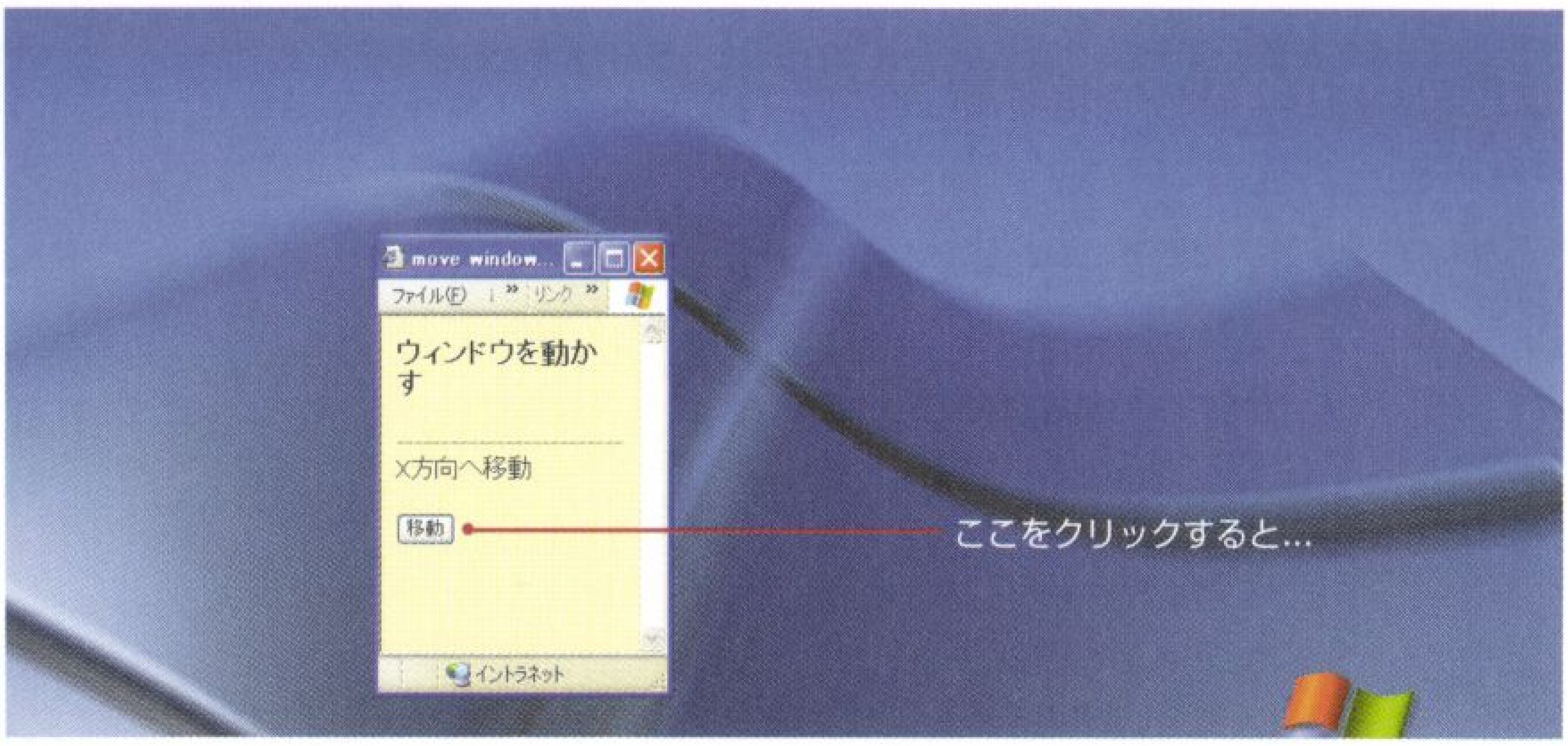
4-10 WIN 4↑ N 4↑ MAC 5↑ N 4↑

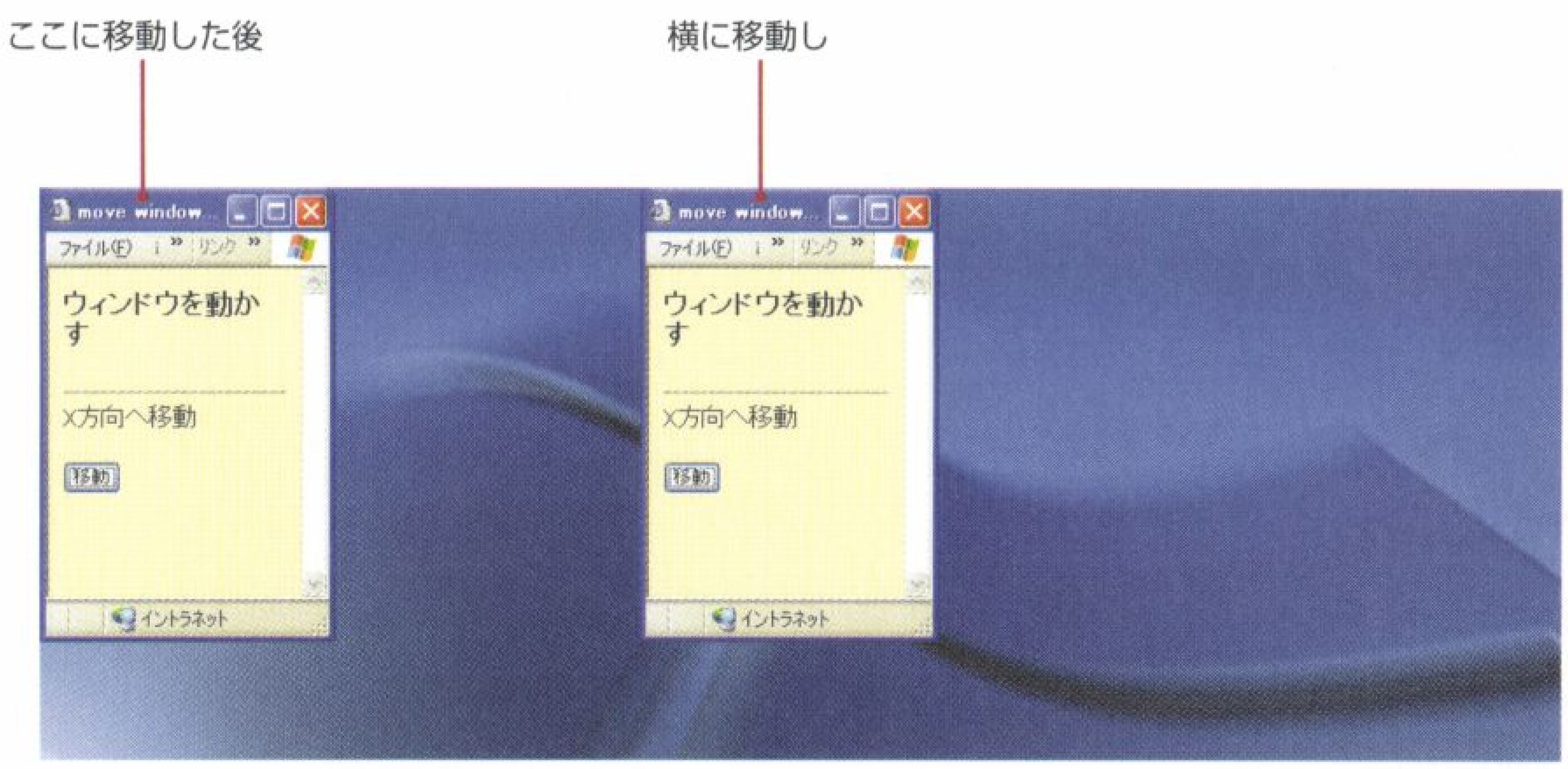
ウィンドウを移動させる

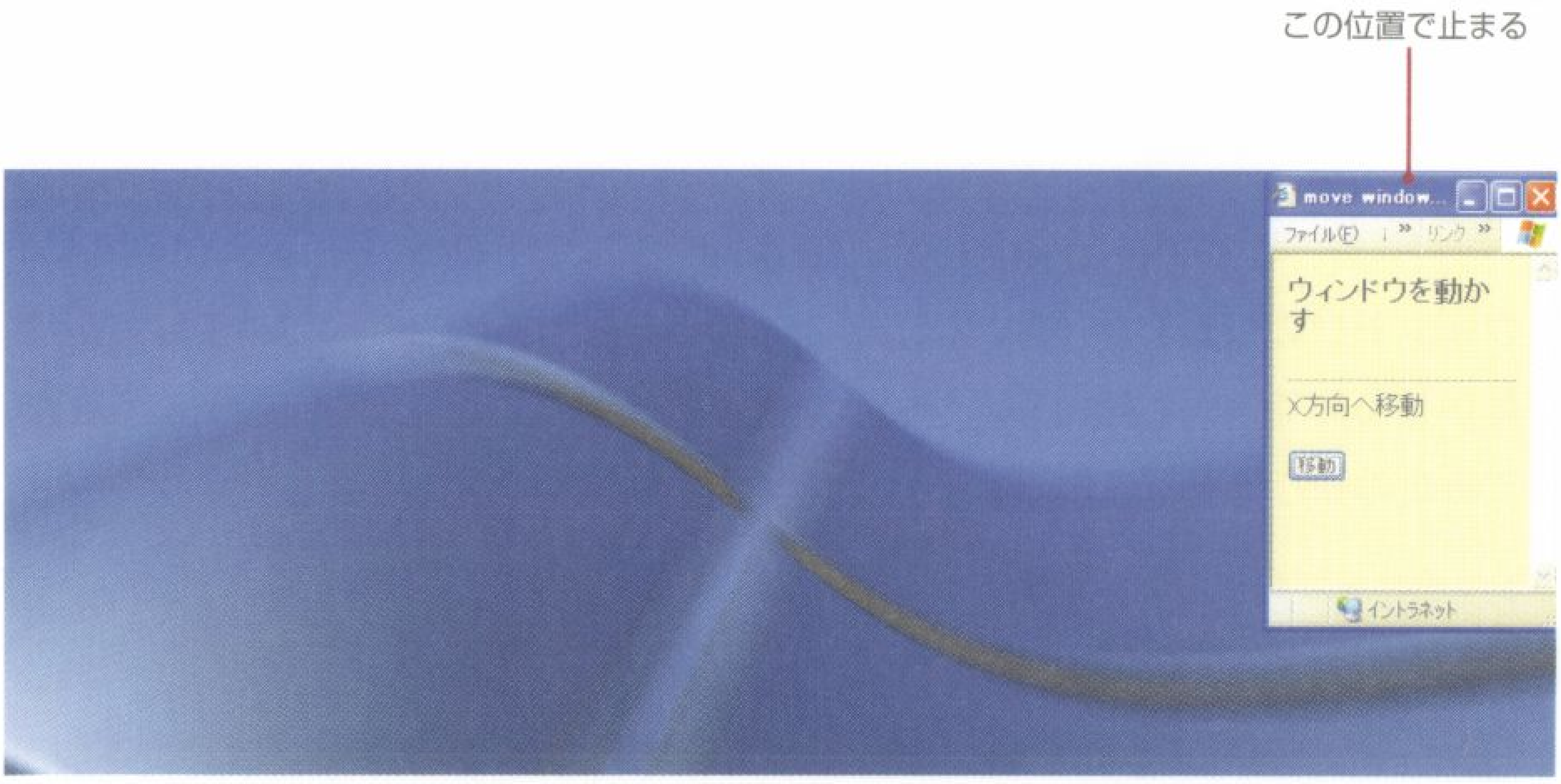
```
変数名 = setTimeout("命令", 呼び出し間隔);
```

ウィンドウの位置やサイズが指定できるなら、IE4以上やNN4ではウィンドウを動かせるのでは、と考える人がいるかもしれません。そこでウィンドウを動かしてみたのがこのサンプルです。

DISPLAY







SOURCE

movewindow2.html

```
<html>
<title>move window</title>

<script language="javascript">
<!--
var displayWidth = screen.availWidth;
var displayHeight = screen.availHeight;
var winWidth = 200;
var winHeight = 300;
var positionX = 0;
var step = 10;
var timerID;
var ua = navigator.userAgent;

function initWindow(){                                              ┐
   if( ua.indexOf("MSIE",0) ){                                      │
      window.resizeTo(winWidth, winHeight);                         │
   }                                                                │
   else if( ua.indexOf("mozilla/4",0) || ua.indexOf                 ├─①
⇒     ("Netscape6",0) ){                                            │
      window.outerWidth = winWidth;                                 │
      window.outerHeight = winHeight;                               │
   }                                                                │
}                                                                   ┘
```

```
function moveWindow(){
    if(positionX < displayWidth-winWidth){ ―――③
        positionX = positionX + step;
        window.moveTo(positionX,0);
        timerID = setTimeout("moveWindow()",10);
    }
    else{
        clearTimeout(timerID); ―――④
    }
}
//-->
</script>

<body onload="initWindow()" bgcolor="#fff8dc">
<h3>ウィンドウを動かす</h3>
<hr />
X方向へ移動
<form>
    <input type="button" value="移動" onclick="moveWindow()" />
</form>                                   ―――②
</body>
</html>
```

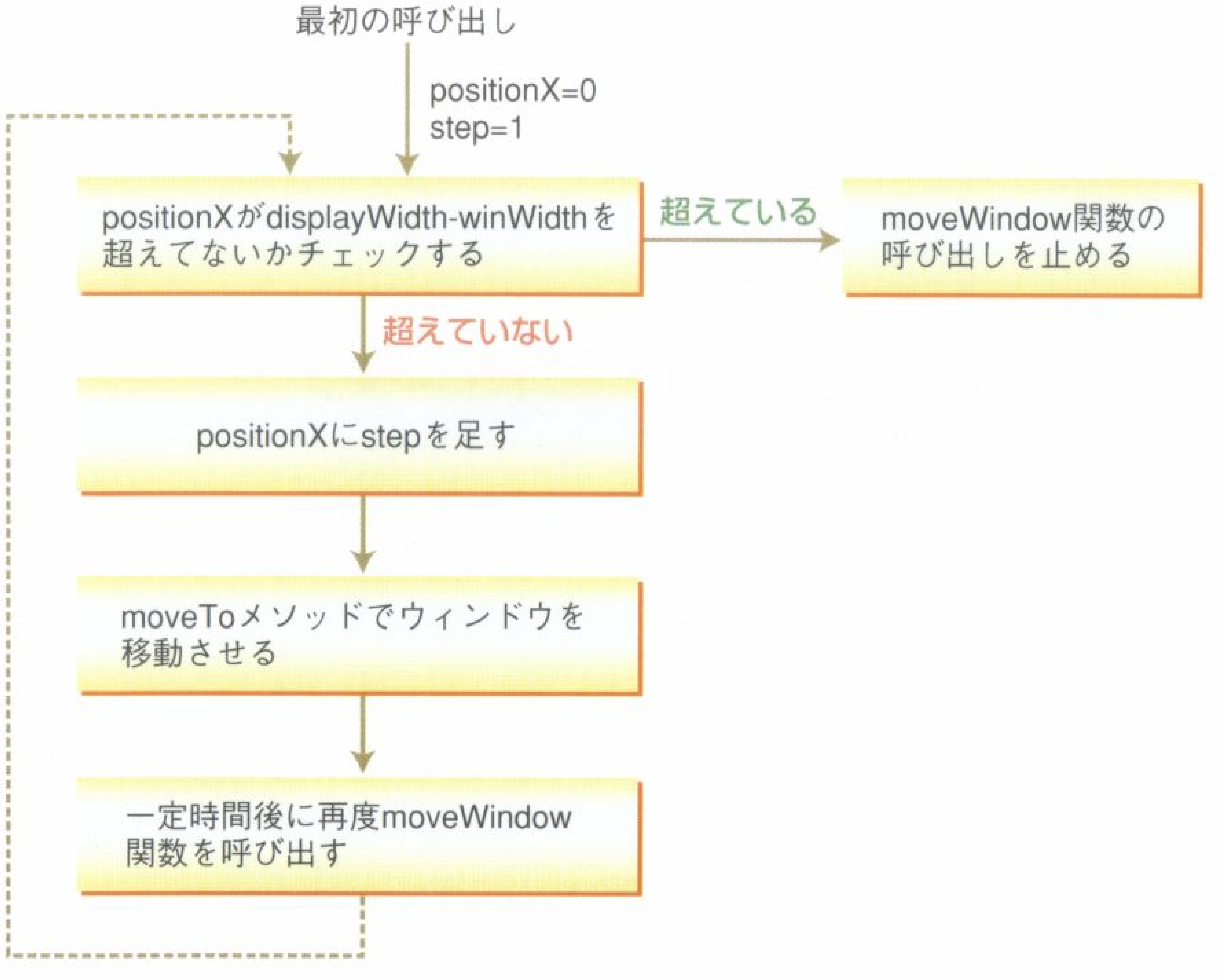

DETAILS

このサンプルでは、Webページがロードされると、ウィンドウのサイズを変更します。また、ページ上に配置された[移動]ボタンをクリックすると、ウィンドウが画面の上端を横方向に移動します。これらの機能を実装するために、ウィンドウサイズを指定するためのinitWindow関数と、ウィンドウを動かすためのmoveWindow関数という2つの関数を作成しています。

initwindow関数は、<body>タグに記述されたonloadイベントハンドラによってWebページロード時に呼び出され、ウィンドウサイズを縦300ピクセル、横200ピクセルに変更します（①の箇所）。このようなウィンドウをリサイズする処理については、P.146で詳細を解説してありますので、そちらを参照してください。

また、moveWindow関数は[移動]ボタンをクリックしたときに発生するonclickイベントハンドラを用いて呼び出されます（②の箇所）。

ウィンドウを動かすには、一定時間ごとにウィンドウの位置を指定し直すという手法を用います。処理のフローは以下のようになります。

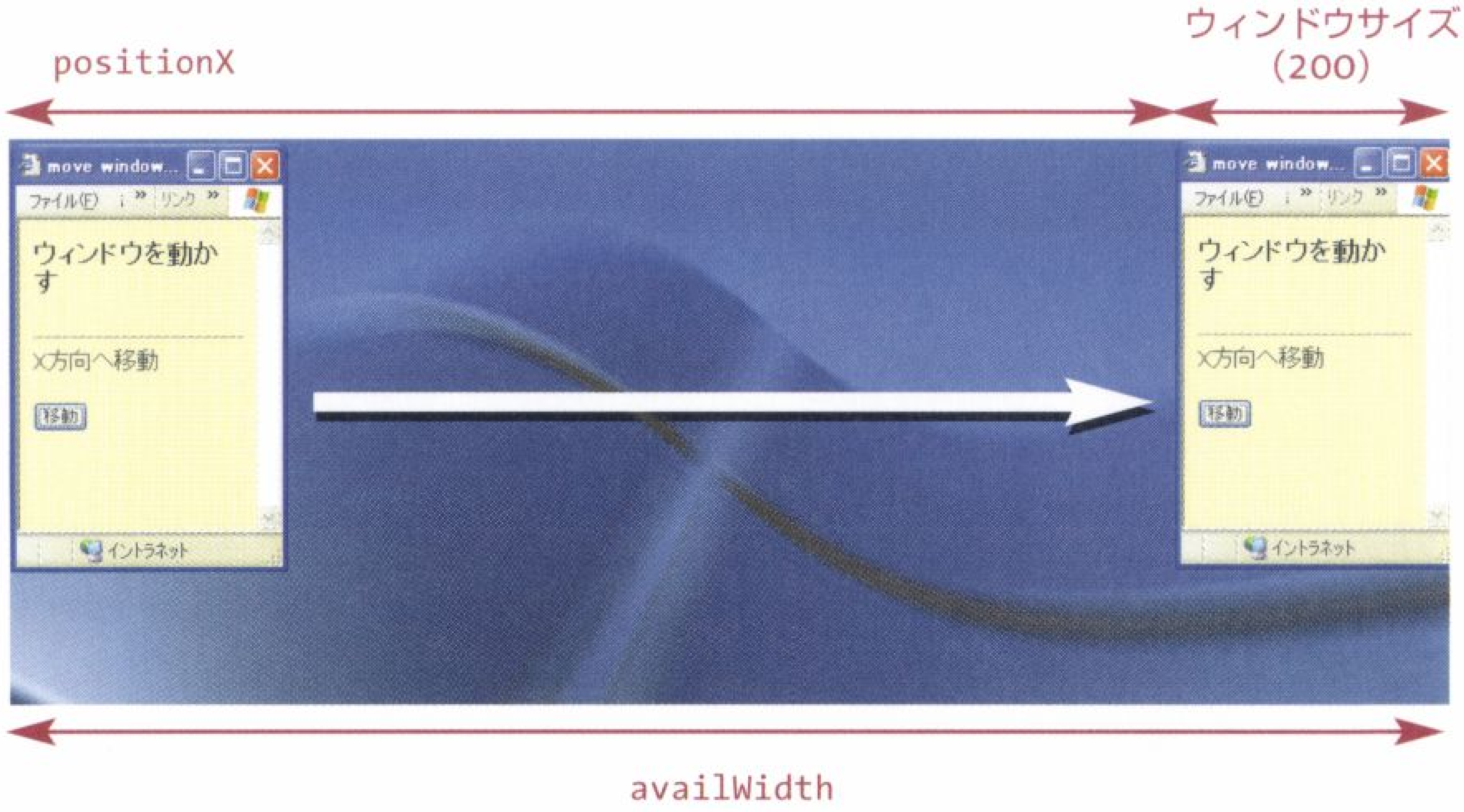


ウィンドウの位置の指定は、P.149で解説したとおりウィンドウの左上端の座標です。ウィンドウを最初に表示するのは画面の左上なので、位置は(0,0)となります。ここからX座標を格納している変数positionXを変化させていくことにより、ウィンドウが動きます。変数positionXは、画面の左上の0からウィンドウが右上にいくまで変化します。このとき変数positionXをどこまで動かすかチェックしなくてはなりません。それを実装しているのが③の部分です。最大値としては、変数displayWidthに格納されているスクリーンの横幅から動かしているウィンドウの幅winWidth(このサンプルでは200ピクセル)を引いたものになります。スクリーンの横幅を調べるために、サンプルではavailWidthプロパティを使用しています(TIPS参照)。

チェックして変数positionXが最大値(displayWidth−winWidth)を超えている場合は、clearTimeoutメソッドを使って再度moveWindow関数を呼び出すのを止めることにより、ウィンドウの移動が止まります(④の箇所)。なお、clearTimeoutメソッドの引数として与えている変数timerIDは、後述するsetTimeoutで設定されているタイマーを識別するための変数です。

変数positionXが最大値を超えていない場合は、現在のウィンドウの位置を表す変数positionXに、移動距離を表す変数stepを足します。これにより新しい

ウィンドウの位置を作ることができたので、moveToメソッドを用いて実際にウィンドウを移動させます。



最後にこの動作を何回も繰り返すために、setTimeoutメソッドで再度moveWindow関数を呼び出します。このとき、ここで設定したタイマーを解除できるように、setTimeoutメソッドからの戻り値を変数timerIDに格納します。この変数がタイマーを識別するための変数となります。setTimeoutメソッドの書式は、以下のようになります。

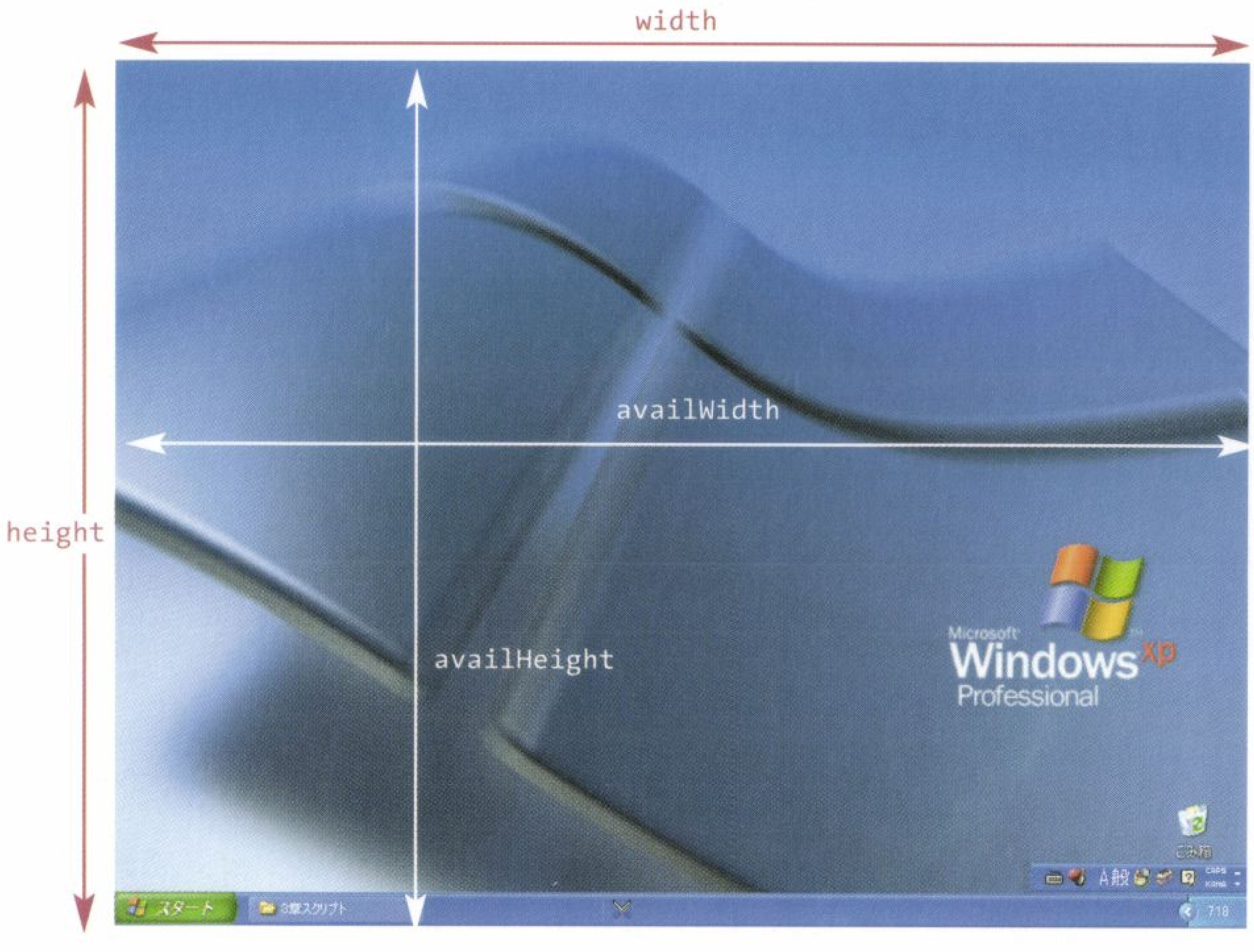
```
変数名 = setTimeout("命令", 呼び出し間隔);
```

今回のスクリプトでは、moveWindow関数を10ミリ秒ごとに呼び出しています。これでウィンドウが移動するというスクリプトを実現することができます。

TIPS

画面の解像度を知るにはいくつかのプロパティがあります。このサンプルではavailHeightプロパティ、availWidthプロパティを用いています。しかし、これ以外にheightプロパティ、widthプロパティがあります。これらの関係は次ページの図のようになります。

availHeightプロパティ、availWidthプロパティはメニューバーやタスクバーを除いた画面の大きさを返します。



COLUMN

制御コード

JavaScriptで文字列を出力するときに、改行やタブはどうやって表現すればよいのでしょうか？　そのときには以下のコードを用います。この本もサンプルでもアラート内に表示する文字列などで「¥n」などを用いています。Unicodeによる指定はIE4以降とNN4.06以降で用いることができます。

制御コード	Unicode	名　前	シンボル
¥b	¥u0008	バックスペース	<BS>
¥t	¥u0009	タブ	<HT>
¥n	¥u000A	ラインフィールド	<LF>
¥f	¥u000C	フォームフィールド	<FF>
¥r	¥u000D	復帰	<CR>
¥"	¥u0022	ダブルクォーテーション	"
¥'	¥u0027	シングルクォーテーション	'
¥¥	¥u005C	円（バックスラッシュ）	¥

4-11 親ウィンドウのコントロール

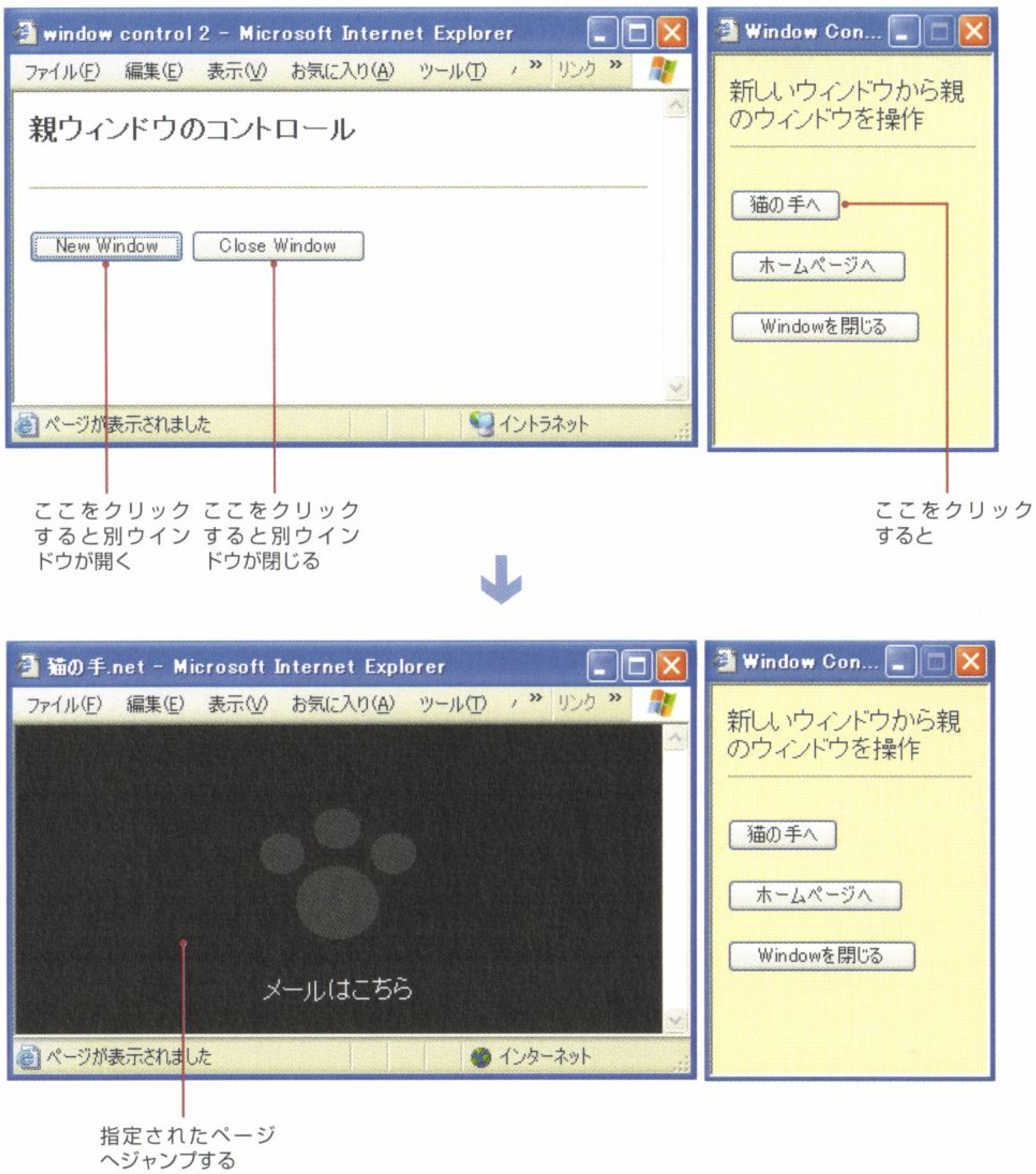
WIN e 4↑ N 4↑ MAC e 4↑ N 4↑

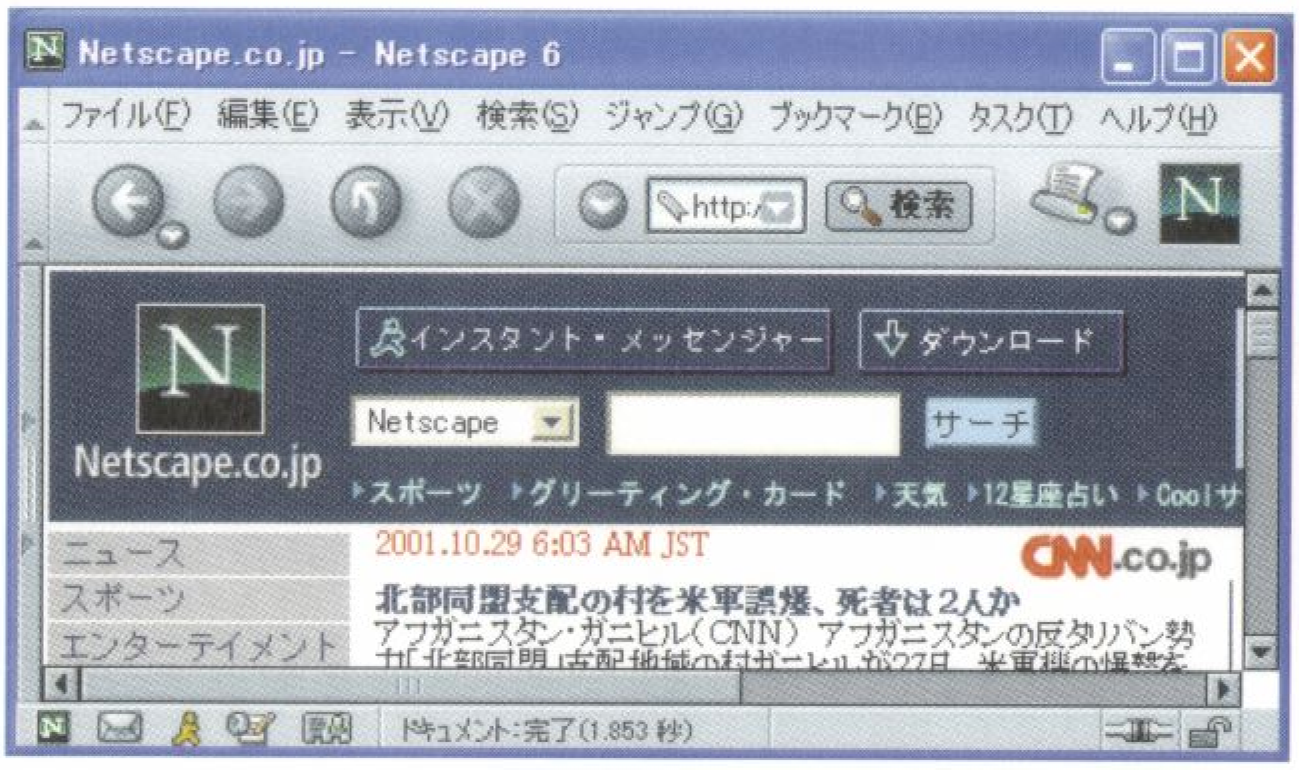
```
window.opener.location.href = "URL"
```

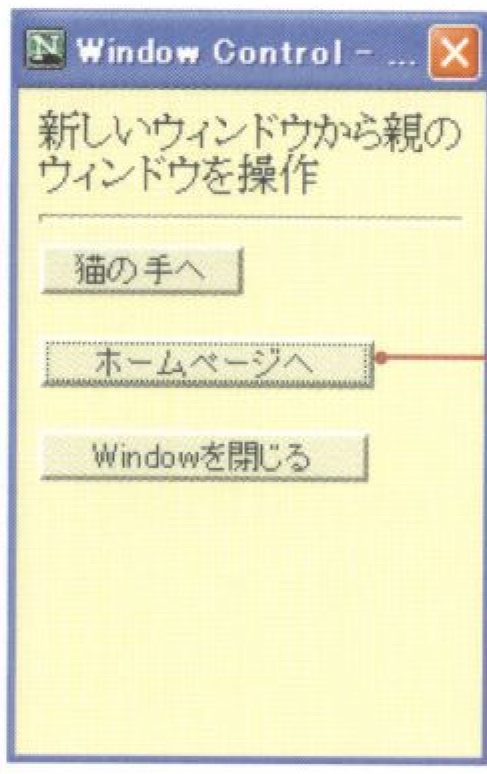
```
window.opener.close();
```

JavaScriptを使用して、新たにウィンドウを開くことができます。開いたウィンドウに対して、さまざまな操作を行うことも可能です。それでは、その新しく開いたウィンドウから元のウィンドウ（親ウィンドウ）をコントロールすることはできるのでしょうか。ここでは、開いた先から開いた側のウィンドウをコントロールするスクリプトを解説します。

DISPLAY







ここをクリックすると「ホームページ」へジャンプするが、NNでしか動作しない

SOURCE　　windowcontrol2.html

```
<html>
<title>window control 2</title>
<script language="javascript">
<!--
var win1;

var win1;
function openWin(){
   win1=window.open('controlwin.html','newwin',
      'toolbar=no,location=no,status=no,menubar=no,
      scrollbars=no,width=180,height=250');
}

function closeWin(){
   if(win1.closed == false){
      win1.close();
   }
   else{
      alert("すでにウィンドウは閉じられています");
   }
}
//-->
</script>
```

```
<body bgcolor="#fff8dc">
<h3>親ウィンドウのコントロール</h3>
<hr />
<form>
<input type="button" value="New Window" onclick="openWin()" />
<input type="button" value="Close Window" onclick="closeWin()"
⇨       />
</form>
</body>
</html>
```

controlwin.html

```
<html>
<title>Window Control</title>
<script language="javascript">
<!--
function goNekonote(){
   window.opener.location.href="http://www.neko-no-te.net/";   ①
}

function goHome(){
   window.opener.home();   ②
}

function closeWin(){
   if(opener.closed == false){
      window.opener.close();   ③
   }
   window.close();

}
//-->
</script>
<body>
新しいウィンドウから親のウィンドウを操作
<hr />
<form>
   <input type="button" value="猫の手へ"
⇨     onclick="goNekonote()" /><br /><br />
```

```
    <input type="button" value="ホームページへ"
➡       onclick="goHome()" /><br /><br />

    <input type="button" value="Windowを閉じる"
➡       onclick="closeWin()" />
</form>
</body>
</html>
```


DETAILS

新たなウィンドウを開いたり閉じたりするwindowcontrol2.htmlは、「4-7 ボタンクリックでウィンドウを開閉する」で紹介したwincontrol.htmlと同じです。ただ、このスクリプトでは新たに開くウィンドウにcontrolwin.htmlというファイルを読み込んでいます。

スクリプトによって開かれる新しいウィンドウ(controlwin.html)には、3つのフォームボタンが配置されており、1つ目は新たなリンクへ、2つ目はWebブラウザで指定したホームページに、そして3つ目のボタンはすべてのウィンドウを閉じるようになっています。ただし、2つ目のホームページに戻るボタンはNN4、NN6でないと使えません。

新しく開かれたウィンドウのボタンは、どれもクリックすると関数を呼び出すように設定されています。一番上のボタンをクリックすると、①の部分でgoNekonote関数が呼び出されます。この関数は、現在のウィンドウを開いた、元のウィンドウのURLを変更します。記述は以下のとおりです。

```
window.opener.location.href = "URL"
```

上記の記述では、現在のウィンドウを表すwindowオブジェクト、次のopenerプロパティは新しいウィンドウを開いた、元のウィンドウ(親ウィンドウ)を示します。さらに、親ウィンドウのlocationオブジェクトのhrefプロパティに、新たなWebページのURL(このサンプル中では「http://www.neko-note.net/」)を代入します。この結果、親ウィンドウに新しいWebページが開かれるのです。

次に2つ目のボタンですが、これはWebブラウザに設定されたデフォルトページに移動するものです。デフォルトページへの移動にはhomeメソッドを用いま

す。このメソッドが動作するのはNN4以降です(②の箇所)。

最後に3つ目のボタンですが、これはウィンドウオブジェクト全体を閉じてしまうというものです。最初に、開いた親ウィンドウを閉じています。以下のような記述になっています。

```
window.opener.close();
```

また親ウィンドウだけでなく、現在のウィンドウも閉じます(③)。

```
window.close();
```

これを行わないと、新しいウィンドウのみが表示されたままになります。また、親ウィンドウだけを自分で閉じてしまったときは、3つのボタンをクリックしたとしても、すでに開いた元のウィンドウは閉じてしまっているのでスクリプトエラーが起こります。

なお、エラーはclosedプロパティを用いて事前に親ウィンドウが閉じているかどうかを確認することにより、避けることができます。

TIPS

ウィンドウを開くときには必ず閉じるための方法を表示するようにしましょう。さまざまなアンケートによると、Webページが勝手に開くのはあまり好印象を持たれないようです。よくやたらとウィンドウを何枚も開くサイトがありますが、間違いなく印象がよくありません。最初からユーザーがどのようにサイト内を動くのかを考えて、Web ページの設計をする必要があります。

どうしようもなくてウィンドウを開く場合には、最小の面積で開き、ちゃんと閉じるためのボタンなりイメージを用意しておくのがよいでしょう。

COLUMN

JavaScriptリソース

JavaScriptを制作するにあたって必要となる情報は、以下のサイトにあります。

●ECMA Script (ECMA-262)

URL▶ http://www.ecma.ch/ecma1/stand/ecma-262.htm

●Microsoft Windows Script Technology

URL▶ http://www.microsoft.com/japan/developer/scripting/（日本語）

URL▶ http://msdn.microsoft.com/scripting/（英語）

●Netscape DevEdge Online - JavaScript Developer Central

URL▶ http://developer.netscape.com/tech/javascript/

●Netscape DevEdge Online - W3C DOM Developer Central

URL▶ http://developer.netscape.com/tech/dom/

●Netscape JavaScript 1.5

URL▶ http://developer.netscape.com/tech/javascript/

●Netscape JavaScript 1.3

URL▶ http://developer.netscape.com/docs/manuals/js/client/jsguide/

●W3C Document Object Model (DOM) Level 1 Specification

URL▶ http://www.w3.org/TR/REC-DOM-Level-1/

●W3C Document Object Model (DOM) Level 2 Core Specification

URL▶ http://www.w3.org/TR/DOM-Level-2-Core/

ボタンクリックでアラートを表示する

alert("表示するテキスト")

　Webページの警告などに用いられる、アラートを表示するスクリプトです。ボタンをクリックすることによりアラートを表示します。

D I S P L A Y



ここをクリックするとアラートボックスが表示される



ここをクリックするとアラートボックスが表示される

S O U R C E　　alert.html

```
<html>
<title>Alert on Click</title>
<script language="javascript">
<!--
   function showMsg(){
      alert("Hello JavaScript!");        ─②
   }
//-->
</script>
<body bgcolor="#fff8dc">
<h3>アラートを表示する</h3>
<hr />
ボタンをクリックするとアラートが表示されます<br />
<form>
   <input type="button" value="Please Click!"      ─①
      onclick="showMsg()" />
</form>
</body>
</html>
```

D E T A I L S

Webページ上に配置されたフォームボタンをクリックすることにより、アラートを表示します。アラートは［OK］ボタンのみが用意されている小さなウィンドウです。

まず、Webページ上には、フォームボタンを配置します(①の箇所)。フォームボタンには、ボタン上に表示される文字列として「Please Click!」をvalue属性により設定します。ここまでは通常のフォームと同じ記述です。この後、アラートを表示させる部分を作成します。上記のスクリプトではアラートを表示する部分をshowMsg関数(②の箇所)として実装しています。そのため、フォームボタンではクリックされたときに、この関数を呼び出すようにします。クリックされたときのイベントハンドラはonclickです。showMsg関数の呼び出しは以下のようになります。

```
onclick="showMsg()"
```

このように記述することで、フォームボタンをクリックするとonclickイベントハンドラで指定された関数が呼び出されます。

アラートを表示させるには、以下のようなメソッドを用います。

```
alert("表示するメッセージ");
```

そのため、showMsg関数内では、上記のalertメソッドを用いています。

なお、サンプルのスクリプトでは、関数を呼び出す形にしてありますが、<input>タグのonclickイベントハンドラに、直接アラートの呼び出しを記述することもできます。

```
onclick="alert('Hello JavaScript!')"
```

この記述は表示するメッセージがあまり長くない場合には、スクリプトも見にくくならず有効です。

T I P S

アラートは、メッセージを伝える手段としては非常に有効です。しかしアラートは、そのアラートを消すまで生成したウィンドウを操作できなくしてしまいます。これはWebページを訪問したユーザーにとっては迷惑な場合もあります。特に、Webページ読み込み時に必ず表示されるようなアラートは基本的には避けるべきでしょう。アラートはフォームの入力内容にエラーがあったときや、操作を確認するときなどに使いましょう。

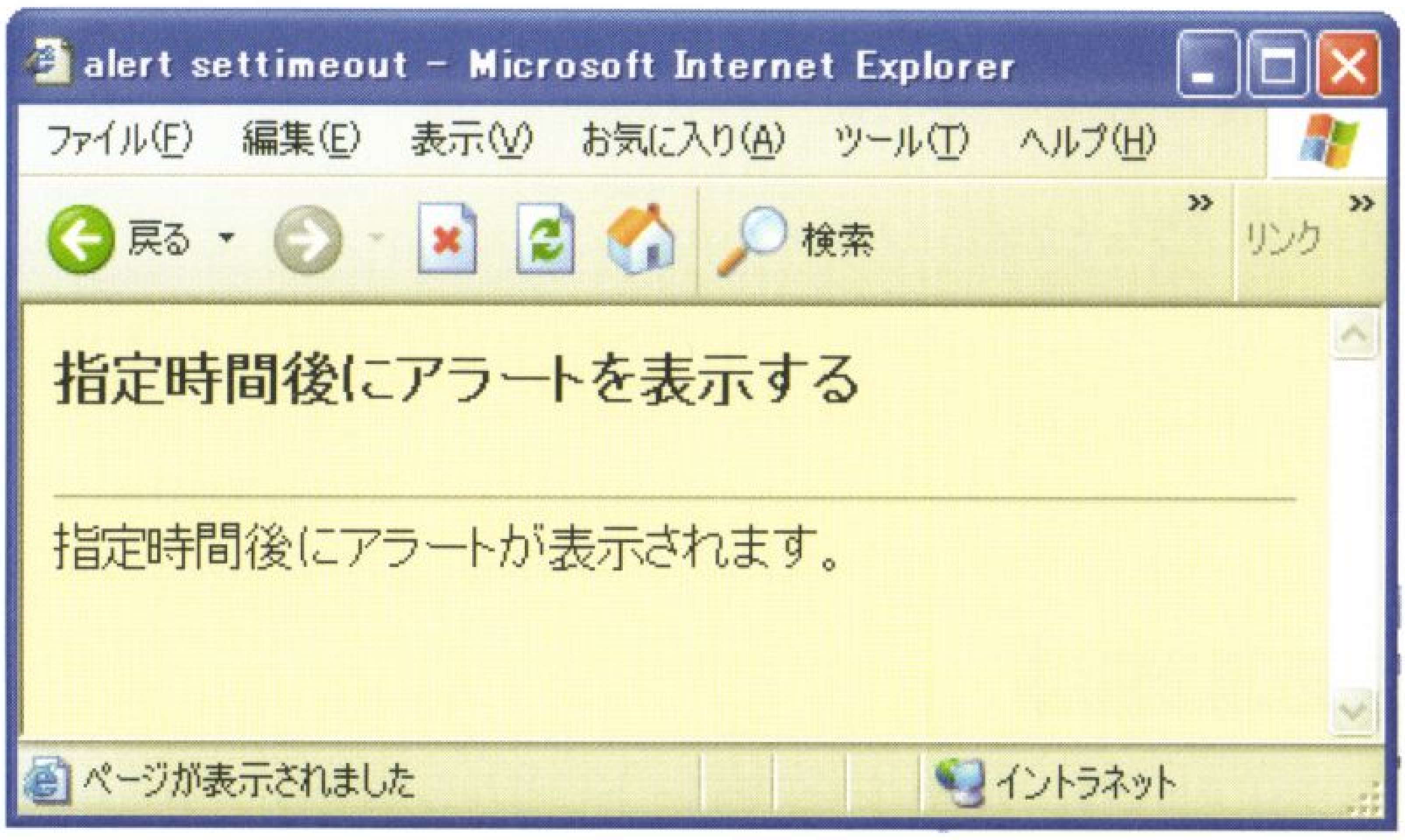
5-2 WIN 4↑ N 4↑ MAC 4↑ N 4↑

指定時間後にアラートを表示する

setTimeout('実行する命令', 時間)

指定した時間が経過するとアクションを起こすスクリプトです。ちょうどタイマーをセットするような感じです。Webページを読み込んだ数秒後に突然メッセージを表示させたり、指定した時間が経過したときに新たなWebページを開いたりすることができます。アニメーションやスライド効果も、タイマーを応用することにより実装できます。

DISPLAY





5秒たつと自動的にアラートボックスが表示される

SOURCE

alertsettimeout.html

```
<html>
<title>alert settimeout</title>
<script language="javascript">
<!--
   function alertTimer(){
      alert("Hello JavaScript!");        ②
   }
//-->
</script>
<body onload="setTimeout('alertTimer()', 5000)"   ①
➡  bgcolor="#fff8dc">
<h3>指定時間後にアラートを表示する</h3>
<hr />
指定時間後にアラートが表示されます。<br />
</body>
</html>
```

DIALOG

DETAILS

このスクリプトでは、Webページをロードしてから5秒後にアラートが表示されます。

Webページ全体が読み込まれたときに関数やメソッドを実行するには、<body>タグのonloadイベントハンドラを用います。ここではonloadイベントハンドラにより、アラートを呼び出すように記述しています。①の部分で、onloadイベントハンドラには以下のような記述が行われています。

```
onload = "setTimeout('alertTimer()', 5000)"
```

この記述で用いられているsetTimeoutメソッドは、一定時間後に命令を実行するメソッドです。書式は以下のようになります。

```
setTimeout('実行する命令', 時間)
```

括弧内の最初には、実行する関数やメソッドを指定します。サンプルでは、

「Hello JavaScript!」というメッセージを表示するためのalertTimer関数を呼び出すように記述しています（②の箇所）。続けてカンマ（,）で区切って、命令を実行するまでの時間を指定します。ここで気を付けるのは、時間の単位が1/1000秒だということです。ここでは「5000」という数値が指定されているので、5秒後にalertTimer関数が呼び出されます。

T I P S

一定時間後に何らかのアクションを起こすというスクリプトや記述は、よくWebページをリロードする際に用いられます。Webページを引っ越した場合や、期間限定で何らかのメッセージを見せたいといった場合です。上記のスクリプトでは一定時間後にアラートを表示させていますが、実際にはlocationプロパティ、またはreplaceプロパティを用いてWebページのURLを変更することが多いでしょう。実際に記述すると以下のような関数になります。

```
function replaceURI(){
  window.location.replace
⇒     ("http://www.neko-no-te.net/");
}
```

なお、HTMLの<meta>タグを用いても、Webページを一定時間後に移動させることはできます。ですからJavaScriptを用いる代わりに、以下の記述にしても一定時間後にWebページを変更することができます。

```
<meta http-equiv="refresh" content="5;
⇒  url=http://www.neko-no-te.net" />
```

＊ 時間は「秒」で指定する

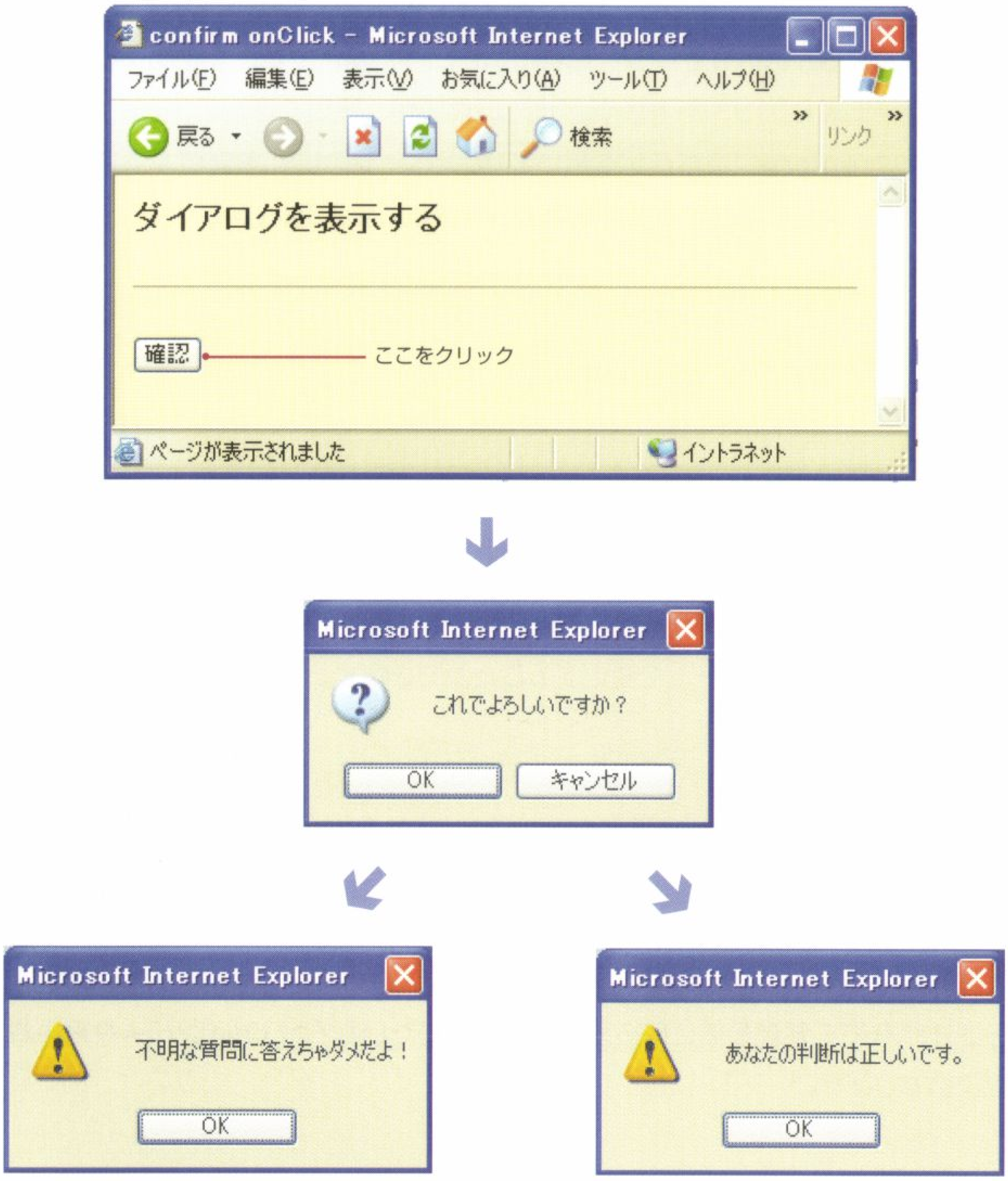
HTMLとJavaScriptの使い分けは、単純にWebページを移動させたいだけならばHTMLで、移動の前に何らかの処理を行いたいならばJavaScriptを用いればよいでしょう。

ダイアログボックスを表示する

```
変数 = confirm("表示するテキスト");
```

確認用のダイアログボックスを出して、ユーザーに確認を求めるスクリプトです。入力されたデータの送信や設定の変更といった、ユーザーの確認をとってから次の作業を行いたい場合に使用しましょう。

DISPLAY



SOURCE confirmonclick.html

```
<html>
<title>confirm onclick</title>
<script language="JavaScript">
<!--
function showMsg(){
   res = confirm("これでよろしいですか？");  ②
   if (res == true)
      alert("不明な質問に答えちゃダメだよ！");  ③
   else
      alert("あなたの判断は正しいです。");
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>ダイアログを表示する</h3>
<hr />
<form>
   <input type="button" value="確認" onclick="showMsg()" />  ①
</form>
</body>
</html>
```

DETAILS

上記のスクリプトでは、「確認」と書かれたフォームボタンをクリックすると、確認を求めるダイアログボックスが表示されます。これに対して［OK］か［キャンセル］のどちらかを選んでクリックすると、選んだ答えによってアラートが表示されます。

フォームのボタンをクリックすると、onclickイベントハンドラにより指定されたshowMsg関数が呼び出されます。この呼び出し部分は、以下のように記述されています(①の箇所)。

```
onclick="showMsg()"
```

ボタンクリックでshowMsg関数が呼び出されると、まずconfirmメソッド

により確認のダイアログボックスが表示されます(②の箇所)。confirmメソッドは以下のような書式です。

```
変数 = confirm("表示するテキスト");
```

「=」の左側の変数には、ダイアログボックスでユーザーに確認をとった答えの値が代入されます。値は、「OK」がクリックされた場合は「true」、「キャンセル」がクリックされた場合は「false」が代入されます。その答えにより、スクリプトの処理を分岐することが可能です。

confirmメソッドの括弧内には、ダイアログボックスに表示するメッセージを記述しておきます。このサンプルでは「これでよろしいですか？」と確認しています。

質問に対して答えると、③の部分のif文が実行されます。[OK]ボタンをクリックした場合には変数「res」にtrueが代入されていますので、「if (res == true)」以下の部分が実行されます。その結果「不明な質問に答えちゃダメだよ！」というアラートの表示が実行されます。逆に[キャンセル]ボタンを選んだ場合には、「あなたの判断は正しいです。」というアラートが表示されます。

TIPS

ダイアログボックスはユーザーに対して、「OK」か「キャンセル」かを確認するものです。そのためダイアログボックスは意思を確認するのに用いるとよいでしょう。たとえば、使用条件や契約条件に同意するかどうかの確認や、購買の意思確認などに用いるとわかりやすいでしょう。

また、表示する文字列は当然ながら「OK」、「キャンセル」で回答できるものにする必要があります。

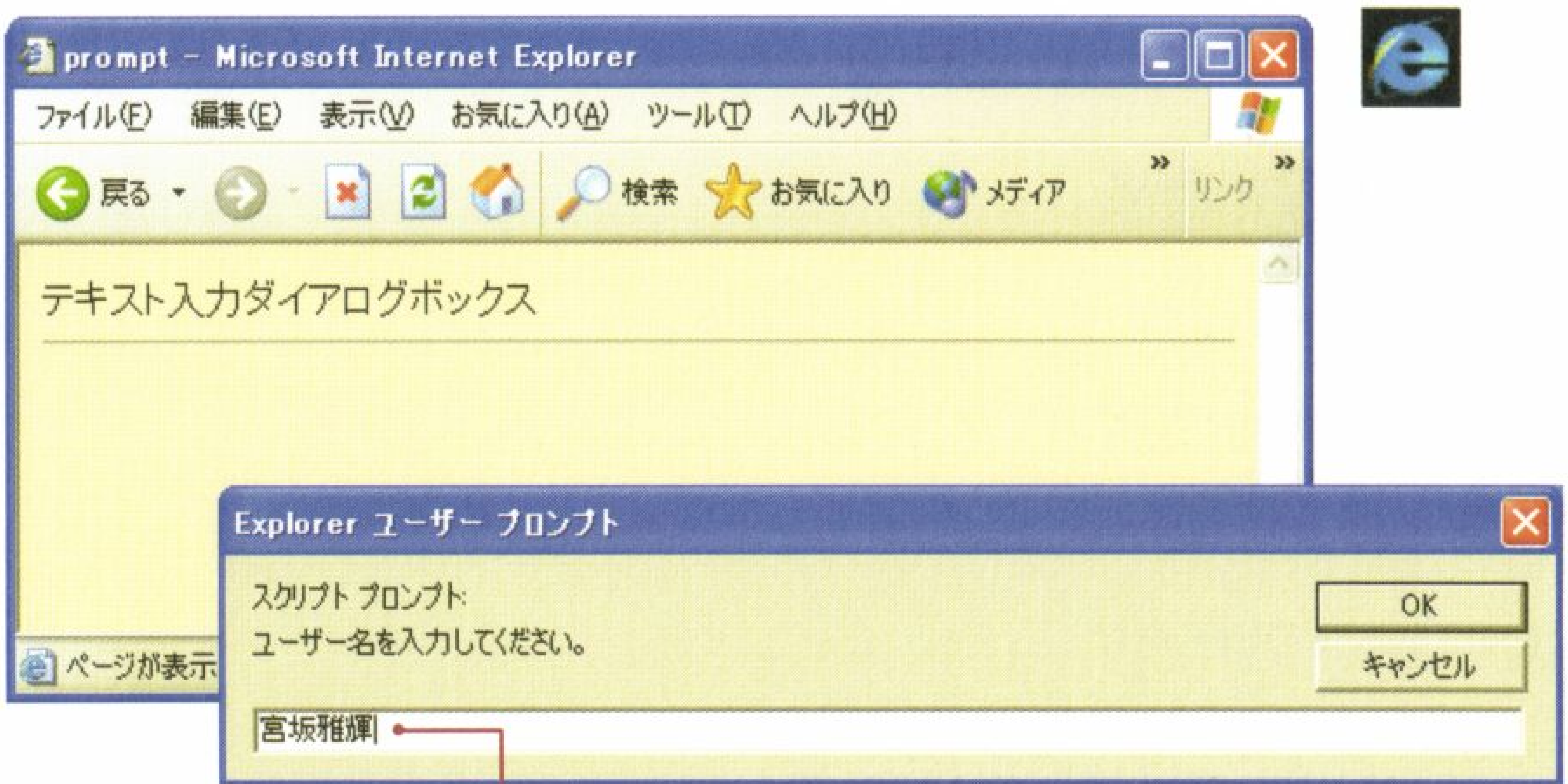
5-4 WIN 4↑ N 4↑ MAC 4↑ N 4↑

プロンプトでユーザーに入力を求め、それをページに反映する

```
変数 = prompt("表示するテキスト","初期値");
```

JavaScriptを使用すると、プロンプト（文字入力用のダイアログボックス）を表示してユーザーに文字列や数字を入力してもらうことが可能です。そして、そこで得た入力結果をスクリプトに反映させることもできます。たとえば、誕生日を入力してもらって星座を割り出したり、ユーザー名を入力してもらったりしたものを、Webページ上に反映させることができます。

D I S P L A Y



自動的にプロンプトが表示されるので、名前を入力して［OK］をクリック



入力された名前がアラートボックスに表示される



プロンプトの形状はWebブラウザによって異なる

SOURCE

prompt1.html

```
<html>
<title>prompt</title>
<script language="JavaScript">
<!--
function welcome(){
    var res;
    res=prompt("ユーザー名を入力してください。","ここに入力");      ②
    alert("ようこそ!¥n" + res + "さん");
}
//-->
</script>
<body onload="welcome()" bgcolor="#fff8dc">      ①
テキスト入力ダイアログボックス
<hr />
</body>
</html>
```

DETAILS

上記のスクリプトではWebページを読み込んだときにプロンプトを表示します。そこに名前を入力して OK をクリックすると入力した名前がアラー

トボックスに表示されます。

Webページが読み込まれたときにスクリプトを起動するためには、<body>タグのonloadイベントハンドラを使用します(①の箇所)。

```
onload = "welcome()"
```

onloadイベントハンドラはWebページの読み込みが終わったときにwelcome関数を呼び出します(②の箇所)。

welcome関数では、まずプロンプトを表示します。プロンプトの書式は以下のようになります。

```
変数 = prompt("表示するテキスト","初期値");
```

最初のダブルクォーテーション内の文字列は、プロンプトに表示されるメッセージです。次のダブルクォーテーション内には、最初にサンプルとして表示させておく文字列や数字を指定しておきます。ユーザーに正しい形式で入力してもらえるように、わかりやすい例を入力しておくとよいでしょう。メッセージや初期値を指定しなくても、プロンプトは実行できます。

ユーザーに入力してもらった値は、変数に代入されます。文字列でも数字でも、どちらでも代入することができます。ここでは変数resに値が代入されます。

最後にユーザーの入力した値を以下の記述により表示します。

```
alert("ようこそ!¥n" + res + "さん");
```

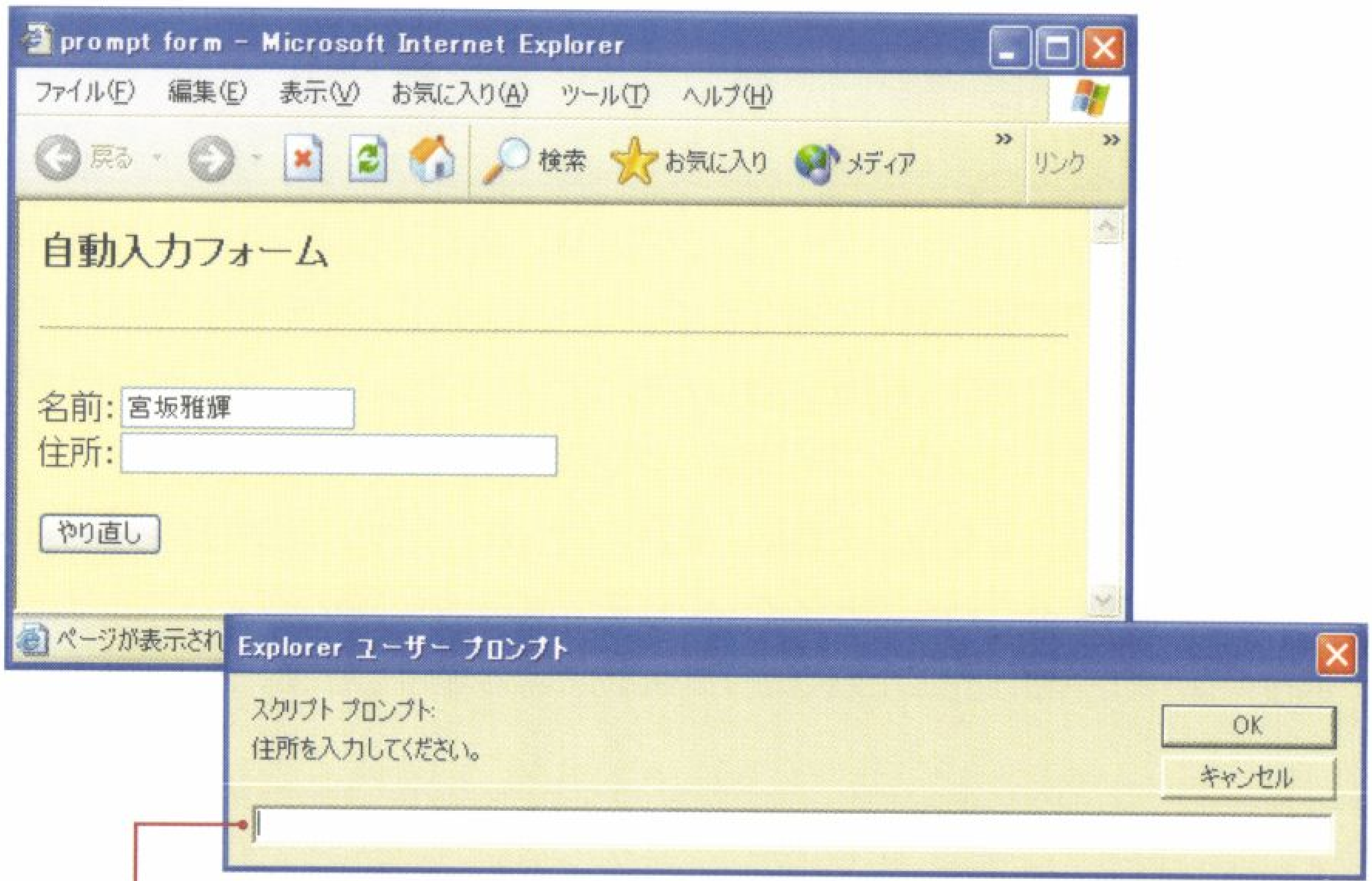
具体的には、アラートに表示するメッセージの一部に変数resの値が使用されています。「+」は数値の加算に使用される演算子ですが、文字列の結合にも使用できます。まず「ようこそ！」という文字列と改行(¥n)に、変数resに代入された値が結合され、その文字列に、さらに「さん」を付け加えた文字列がアラートに表示されます。なお、このスクリプトでは、入力された内容の確認を行っていないので、何も入力されないと、"null"が変数resに格納されてしまうため、「ようこそnullさん」と表示されてしまいます。

プロンプトでユーザーに入力を求め、フォームに反映する

document.formオブジェクト名.elementオブジェクト名.value=値

　フォームを使っていくつもの項目を入力してもらう場合、重要な質問を見落としてしまう場合があります。またどうしても入力してほしい項目がある場合には、目立たせたい場合があります。このスクリプトでは、フォーム入力のWebページがロードされると同時にプロンプトが表示され、指示にしたがっていくだけで、自動的にフォームへの入力が完了します。

DISPLAY



ここに入力された文字列が、自動的に
フォームに入力される

SOURCE

prompt2.html

```
<html>
<title>prompt form</title>
<script language="javascript">
<!--
function inputPrompt(){
```

```
    var res="";
    for(;;){
        res=prompt("名前を入力してください。","");
        if(res=="" || res==null){
            alert("名前が入力されていません");
        }
        else{
            document.myForm.input1.value=res;  ――③
            break;
        }
    }
    res = "";
    for(;;){
        res=prompt("住所を入力してください。","");
        if(res=="" || res==null){
            alert("住所が入力されていません");
        }
        else{
            document.myForm.input2.value=res;
            break;
        }
    }
}
//-->
</script>
<body onload="inputPrompt()" bgcolor="#fff8dc">
<h3>自動入力フォーム</h3>
<hr />
<form name="myForm">
➡  名前：<input type="text" size="20" name="input1" />  ┐
➡  <br />住所：<input type="text" size="40" name="input2" />  ┘①
➡  <br /><br /><input type="button" value="やり直し"
➡  onclick="inputPrompt()" />  ――②
</form>
</body>
</html>
```

DIALOG

DETAILS

このサンプルでは、2つのテキストボックスと「やり直し」と書かれたボタンのあるフォームを配置しています。name属性により、フォームには「myForm」、テキストボックスには、それぞれ「input1」、「input2」という名前が付けられており(①の箇所)、これらはそれぞれformオブジェクト、elementsオブジェクトとして扱われます。

Webページが読み込まれると、②の部分の<body>タグのonloadイベントハンドラによってinputPrompt関数が呼び出されます。

inputPrompt関数では、プロンプトを表示し、そこに入力された値をフォームに代入する処理を行います。この関数内ではさらに、プロンプトに何らかの値が入らない場合にはユーザーに対してアラートを表示し、再度入力を促す処理を行っています。この処理には無限ループを使い、あまり複雑にならないようにしています。

```
for(;;){
    res=prompt("名前を入力してください。","");
    if(res=="" || res==null){
        alert("名前が入力されていません");
    }
    else{
        document.myForm.input1.value=res;
        break;
    }
}
```

無限ループ
変数resの確認
フォームへの代入
ループから抜ける

inputPrompt関数では、まず無限ループに入り、その中でプロンプトの表示を行っています。

```
res=prompt("名前を入力してください。","");
```

上記の記述により「名前を入力してください」というプロンプトが表示され、ユーザーが入力した文字列は変数「res」に代入されます。[キャンセル]ボタンがクリックされた場合は、変数resにnullが代入されます。次に変数resに入力された値を調べます。

```
if(res=="" || res==null)
```

上記のif文では、変数resに空白、またはnullが代入されていないかどうかを調べています。

変数resの値が空白、またはnullの場合にはアラートを表示してif文を抜けます。この場合は無限ループの中なので再度forループの先頭に処理が戻り、プロンプトが表示されます。

また、変数resが空白、またはnull以外の場合には、③の記述のようにフォームに値が代入されます。

```
document.myForm.input1.value = res;
```

この部分で、変数resに代入されている文字列が、myFormというフォーム上にあるinput1という名前のテキストボックスに入力されます。テキストボックスに入力される値はvalueプロパティで設定できます。住所を入力してもらう部分についても、同様の処理を繰り返します。

フォームの一番下にある［やり直し］ボタンをクリックすると、このフォームのonclickイベントハンドラによりinputPrompt関数が呼び出され、再び最初から処理がやり直されます。

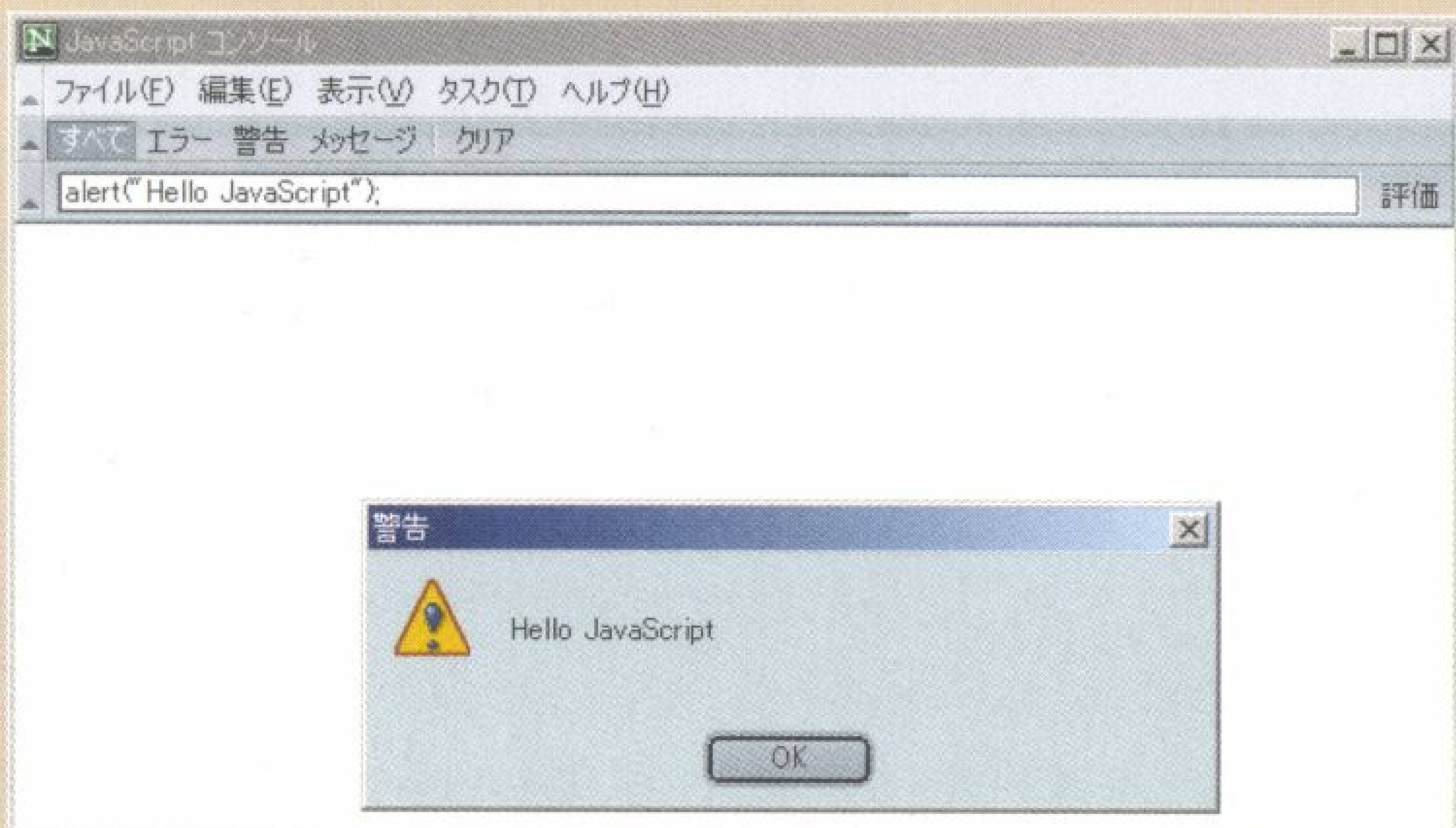
ここでは、2つのテキストボックスに入力してもらうだけなので、あまり役立つとは感じられないかもしれません。しかし、多数のテキストボックスに入力してもらう場合などに、1つ1つの回答を確かめながら対話形式でアンケートを進めていくなど、工夫次第で面白いWebページが作成できるでしょう。

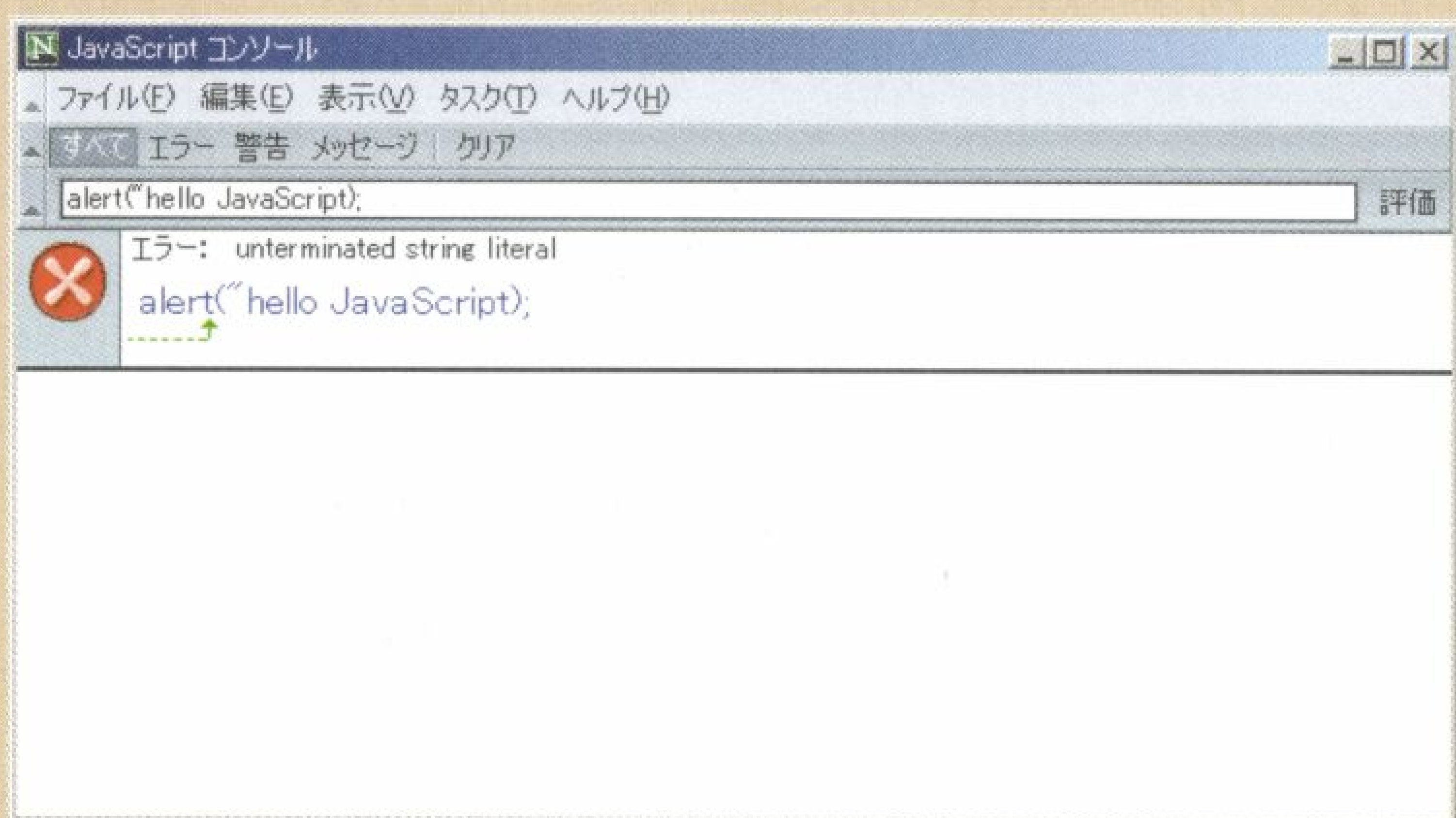
COLUMN

Netscape Navigator内蔵のデバッガ

NNの場所ツールバー（locationツールバー）に「javascript:」と入力すると、内蔵のデバッガ（JavaScriptコンソール）を起動することができます。このとき、「javascript」のキーワードの後ろに付くのはセミコロン「;」ではなくコロン「:」です。

起動したウィンドウは一番上に入力のフォームが、その下に結果の表示領域があります。この入力フォームにJavaScriptの命令を記述して「評価」をクリックすると、下の表示領域に、命令を処理した結果が表示されます。なお、エラーがある場合にはこの表示領域にエラーの内容が表示されます。





なお、Webページを閲覧中にエラーが発生したときに、locationツールバーに「javascript:」と入力しデバッガを立ち上げると、そのときのエラーの内容が表示されます。

JavaScript
Handbook
3rd edition

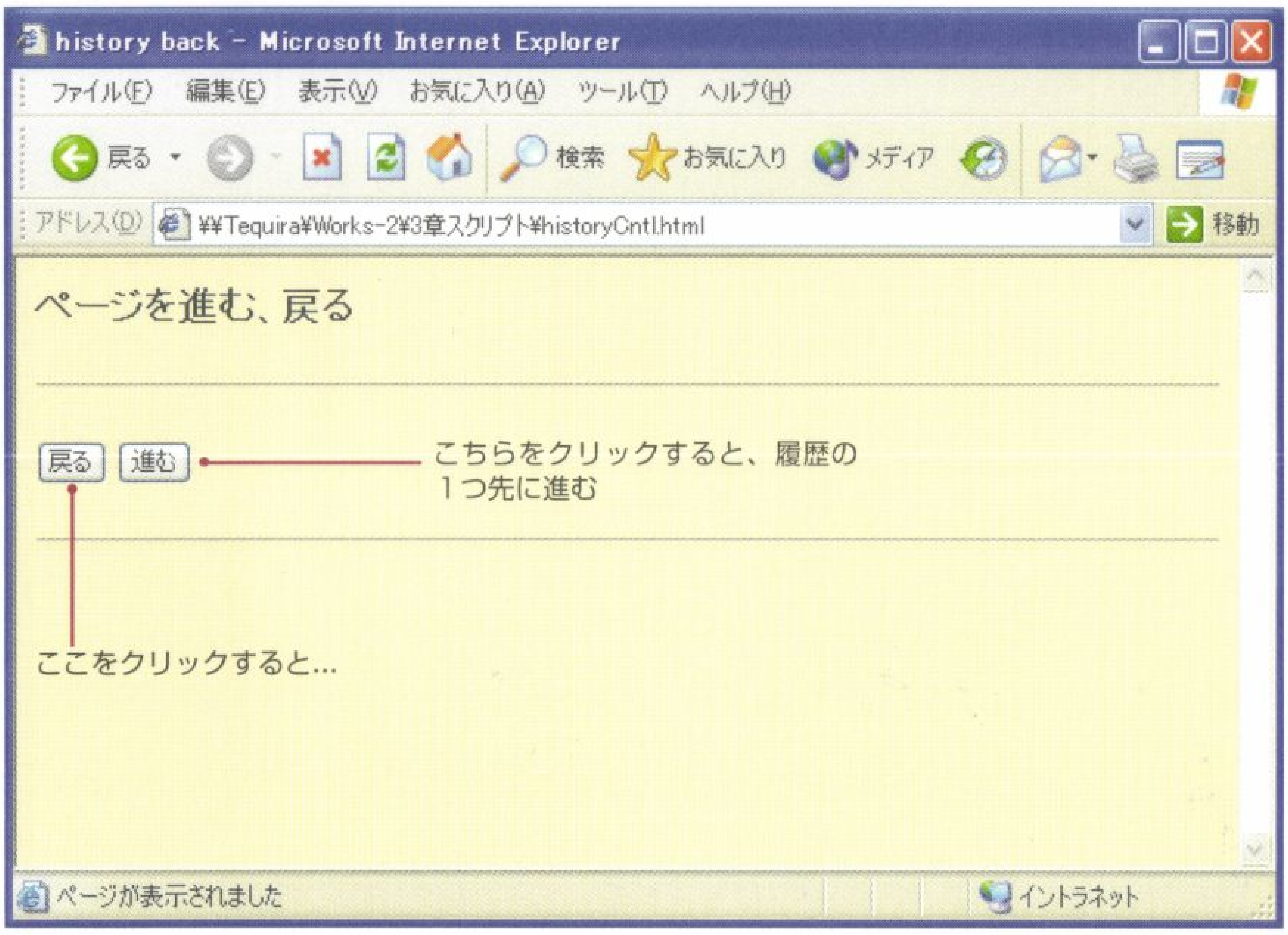
履歴を用いてページを戻る・進む

```
history.back();
```

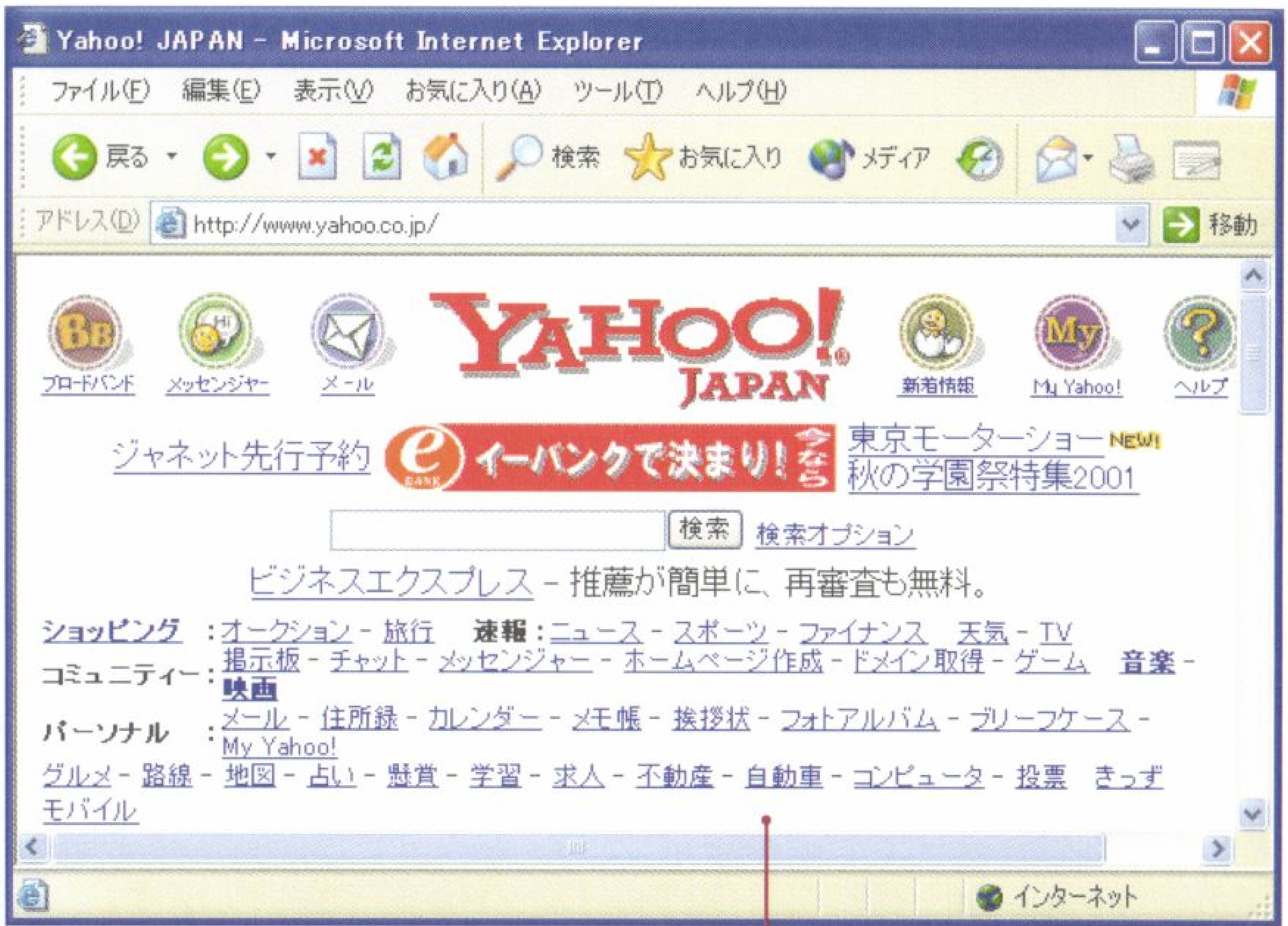
```
history.forward();
```

少し前に表示したWebページをもう一度開きたいとき、Webブラウザのツールバーにある「戻る」ボタンでWebページを戻ることはよくあります。Webページ作成者が何らかの意図を持って、Webページの履歴にある1つ前のページに戻したいときに、次のようなスクリプトを使用します。たとえば、Webページを読み込んだときにパスワード入力画面を表示させ、パスワードが間違っていたらリンクする前のWebページに戻すといったように使用されます。また、逆にWebページを１つ進めたい場合もあります。なお最初にこのWebページを開いた場合は、履歴がないので戻ったり、進んだりすることはできません。

DISPLAY



↓



履歴の１つ前のページに戻る

S O U R C E histrycnt1.html

```
<html>
<title>history back</title>
<script language="javascript">
<!--
function pageBack(){          ①
    history.back();
}
function pageForward(){       ②
    history.forward();
}

//-->
</script>
<body bgcolor="#fff8dc">
<h3>ページを進む、戻る</h3>
<hr />
<form>
    <input type="button" value="戻る" onclick="pageBack()" />
```

```
    <input type="button" value="進む" onclick="pageForward()" />
</form>
<hr />
</body>
</html>
```

D E T A I L S

このサンプル上には2つのフォームボタンが配置されています。フォームボタンで［戻る］ボタンをクリックすると<input>タグに指定されたonclickイベントハンドラによりpageBack関数が、［進む］ボタンをクリックすると同様にpageForward関数が呼び出されます。

①の部分のpageBack関数では、以下のメソッドが実行されています。

```
history.back();
```

historyオブジェクトのbackメソッドは、現在表示されているWebページを、履歴に登録されている1つ前のWebページに戻すというものです。これにより、現在表示されているWebページを見る１つ前のWebページに戻ります。

また、②の部分のpageForward関数では、以下のメソッドが実行されています。

```
history.forward();
```

historyオブジェクトのforwardメソッドは、履歴の1つ先のWebページに進ませます。

このサンプルではフォームボタンを使用していましたが、アンカー内に実装することもできます。

```
<a href="javascript:history.back()">戻る</a>
```

href属性内でhistoryオブジェクトのbackメソッドを実行しています。アンカーをクリックしてJavaScriptを実行させるには、<a>タグのhref属性に命令を指定する必要があります。その際、実行する命令の前に「javascript:」と書くことを忘れないでください。

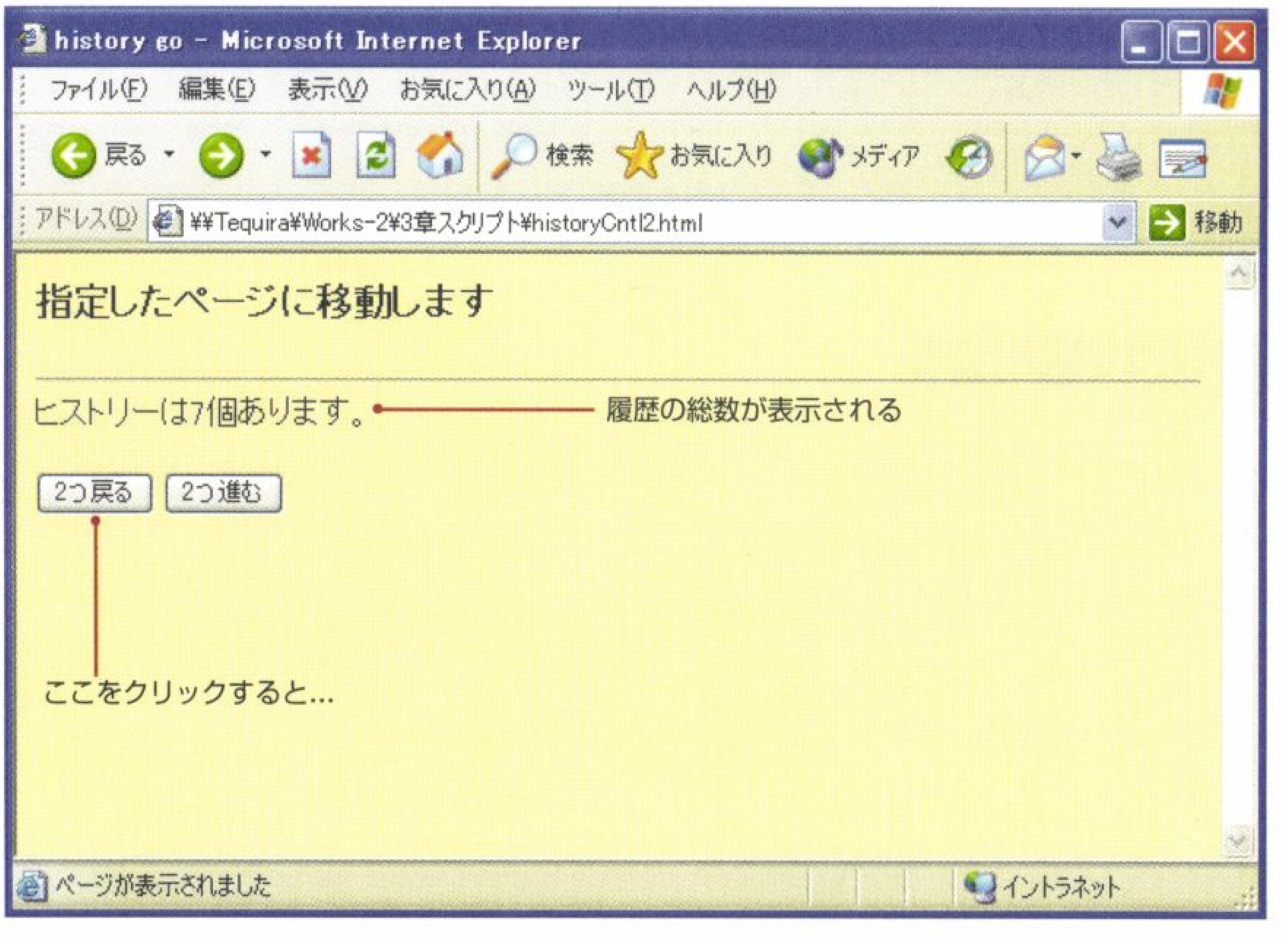
履歴を用いてページを移動する

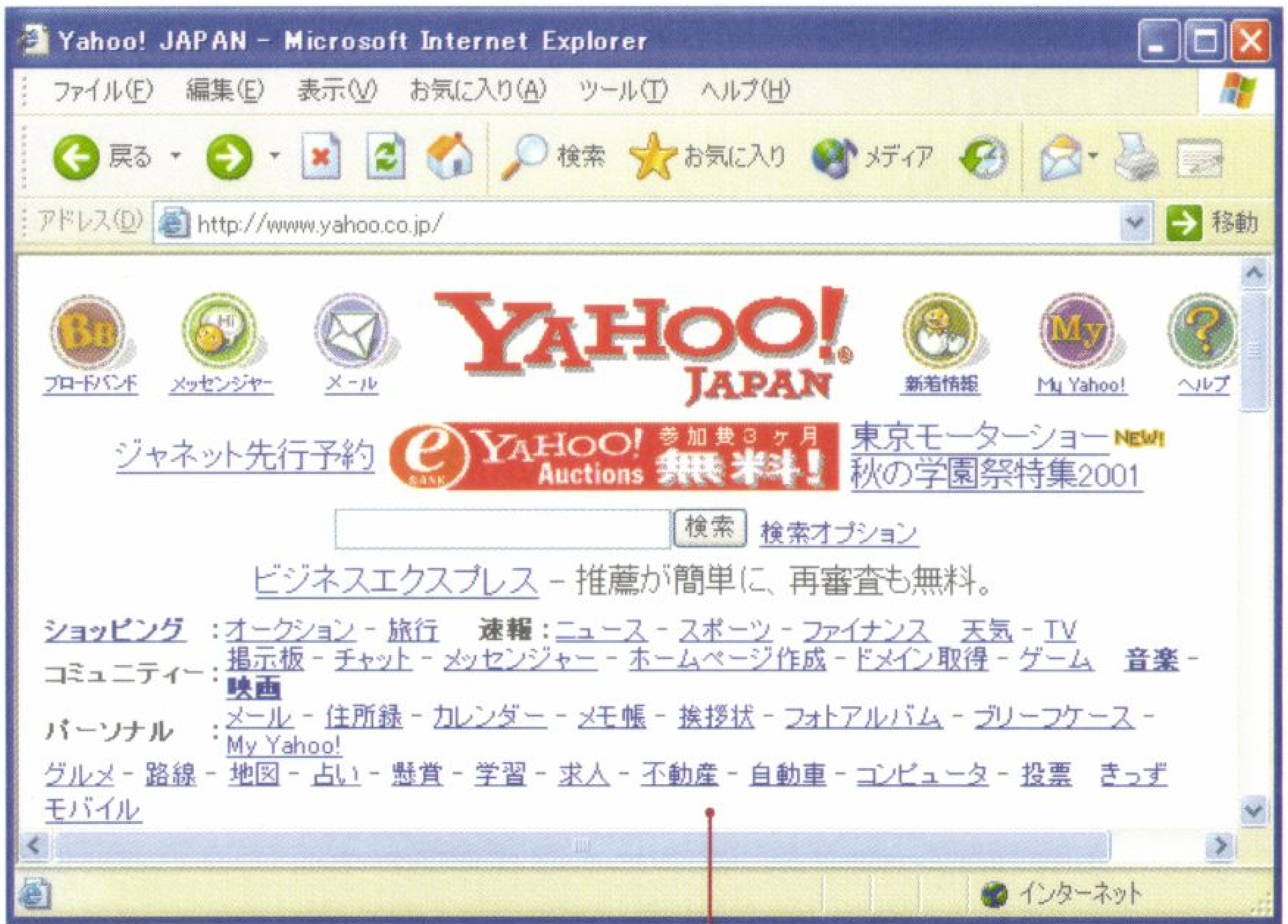
history.length

history.go(移動するページ数)

JavaScriptでは、ただ単に履歴の1つ前や1つ後ろのWebページに移動するだけでなく、いくつ進ませるかも指定することが可能です。とはいえ、指定した数以上の履歴がなければ何も起こりませんので、history.backメソッドhistory.forwardメソッドと組み合わせてスクリプトを書くといいでしょう。

DISPLAY





2つ前の履歴にさかのぼって
ページを表示する

SOURCE historycnt2.html

```
<html>
<title>history go</title>
<script language="javascript">
<!--
   var historyLength = history.length;   ③

function goHistory(pointer){
   history.go(pointer);   ①
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>指定したページに移動します</h3>
<hr />
<script language="javascript">
<!--
   document.write("ヒストリーは" + historyLength +
⇒      "個あります。<br />");   ④
//-->
```

```
</script>
<form>
    <input type="button" value="2つ戻る"
⇒       onclick="goHistory(-2)"/>                    ②
    <input type="button" value="2つ進む"
⇒       onclick="goHistory(2)" />
</form>
</body>
</html>
```

DETAILS

このサンプルでは、2つのフォームボタンが配置されています。[2つ戻る]ボタンをクリックすると、2つ前のWebページに戻り、[2つ進む]ボタンをクリックすると、2つ先のWebページに進みます。これらのボタンをクリックした際に呼び出されるのはgoHistory関数です。

このgoHistory関数は、historyオブジェクトのgoメソッドを実行しています(①の箇所)。goメソッドの書式は以下のようになります。

```
history.go(移動するページ数)
```

移動するページ数にマイナスの数値を与えた場合には履歴を戻り、プラスの数値を与えた場合には履歴を進みます。

ここでgoHistory関数を見てみると、引数で渡された値pointerを移動するページ数の部分に設定しています。そのため、goHistory関数を呼び出している部分の数値が、goメソッドに渡されることになります。[2つ戻る]ボタンに設定されている部分では「-2」を、[2つ進む]ボタンに設定されている部分では「2」を設定しているため、これらの数値がgoメソッドに渡されることになるので、履歴を戻ったり、進んだりすることになります(②の箇所)。履歴がなかった場合は、何も起こらないだけでエラーにはなりません。

またボタンの上には履歴の総数が表示されますが、これにはhistoryオブジェクトのlengthプロパティを使用しています。具体的には③の部分でlengthプロパティの値を変数historyLengthに代入し、それをdocument.writeメソッドで表示させています(④の箇所)。

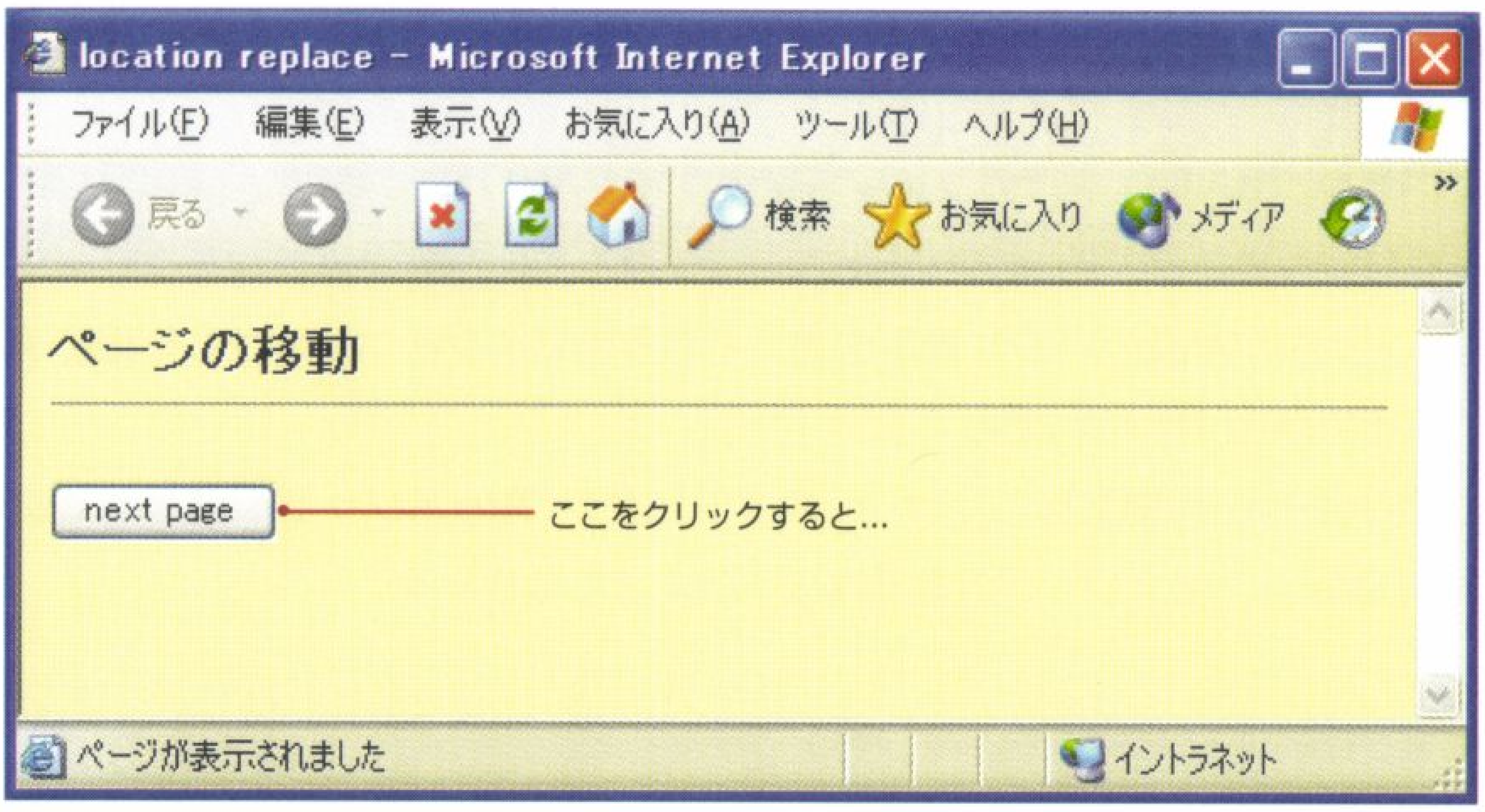
6-3 WIN 4↑ N 4↑ MAC 4↑ N 4↑

指定したページに移動する

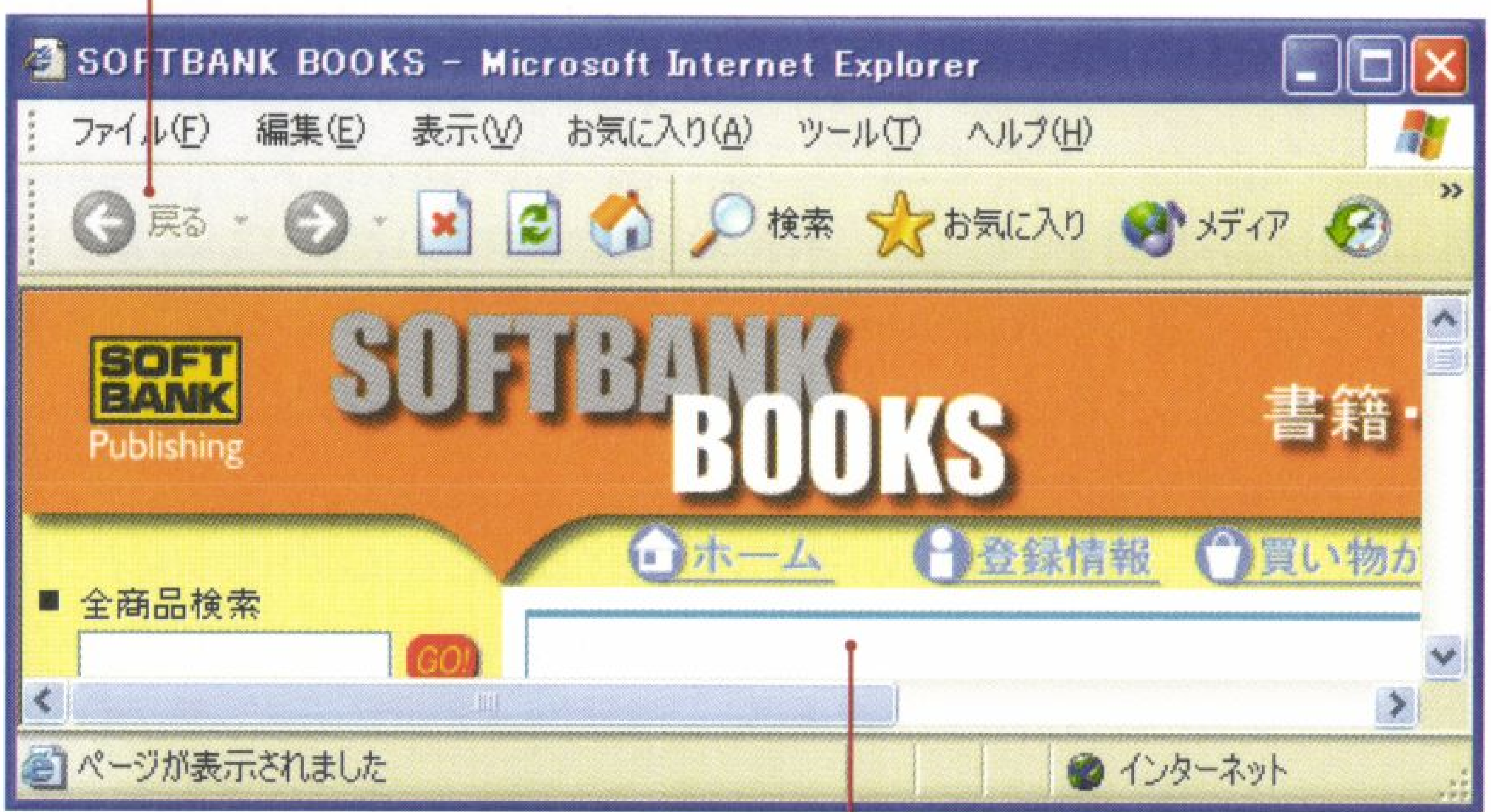
```
windowオブジェクト名.location.replace("URL");
```

通常Webページを移動すると、Webブラウザのツールバーにある「戻る」ボタンで前のWebページに戻ることが可能です。ところが、このサンプルページにあるボタンをクリックしてWebページを移動すると、ボタンの配置してあったWebページには「戻る」ボタンで戻ってくることができなくなります。

D I S P L A Y



戻るボタンは使えない



指定したページに入れ替わる

SOURCE replace.html

```
<html>
<title>location replace</title>
<script language="javascript">
<!--
   function replacePage(){
      window.location.replace("http://books.softbank.co.jp");
   }
//-->
</script>
<body bgcolor="#fff8dc">
<h3>ページの移動</3h>
<hr />
<form>
   <input type="button" value="next page" onclick=
⇒     "replacePage()" />
</form>
</body>
</html>
```

DETAILS

Webページには、「next page」と書かれたフォームボタンが配置されています。クリックすると、<input>タグのonclickイベントハンドラによりreplacePage関数が呼び出されます。

replacePage関数で実行されるのは、以下の命令です。

```
window.location.replace("http://books.softbank.co.jp");
```

これは、windowオブジェクトに対し、locationオブジェクトのreplaceメソッドを使用しています。

replaceメソッドは、現在のWebページのURLを、括弧内に指定したURLに置き換えるメソッドです。その結果、Webページは現在のWebページからreplaceで指定したWebページへと変更されますが、現在のWebページのURLは履歴に残りません。ですから、Webブラウザのツールバーの「戻る」ボタンで戻ろうとしても、戻れなくなってしまうのです。

6-4

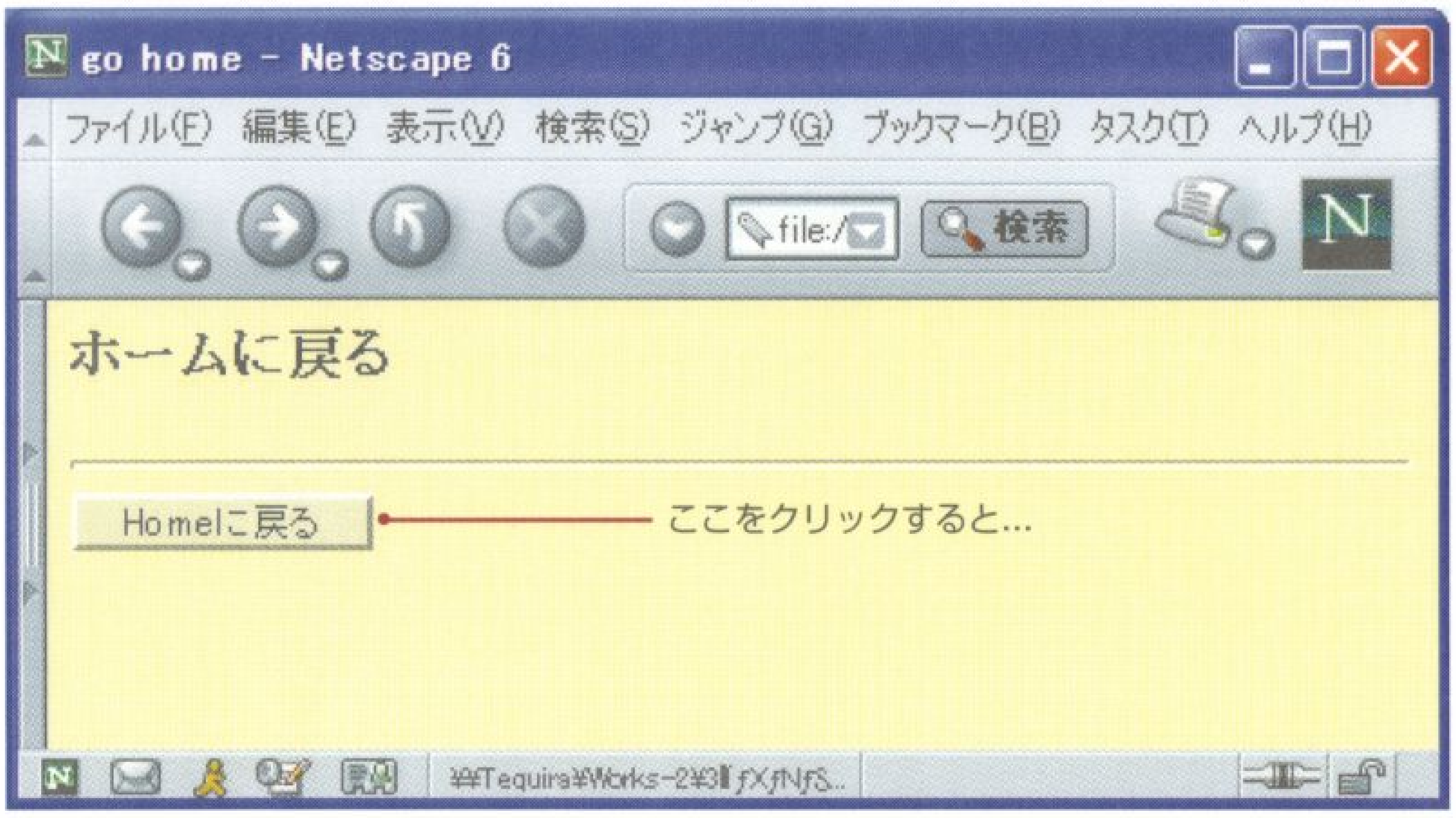
WIN MAC

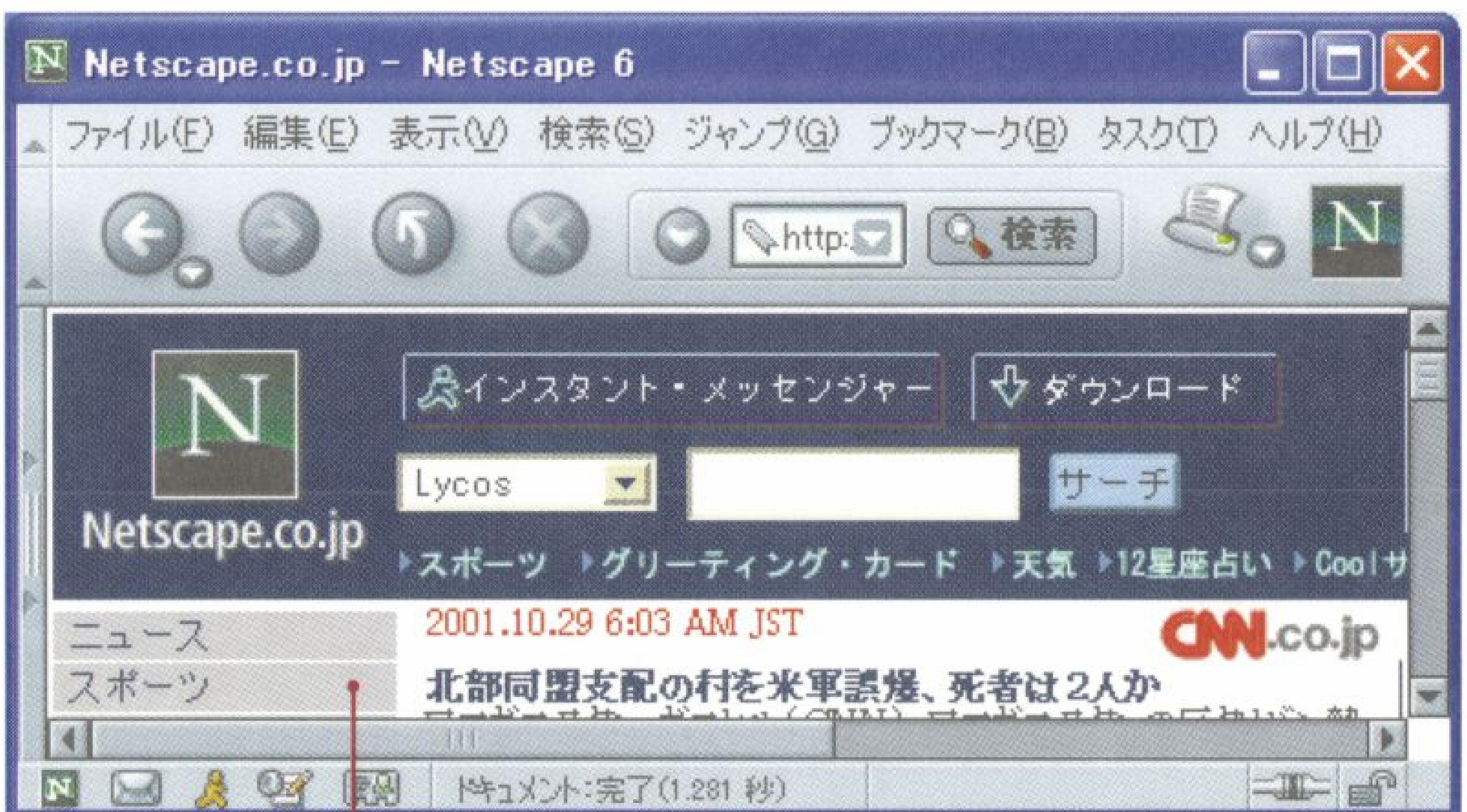
Webブラウザで指定したホームに戻る

JavaScript1.2では、ユーザーの開いているWebページをホームに戻すことができます。ホームとは、ブラウザを起動したときに最初に表示されるWebページのことです。もっともよく開くWebページや、サーチエンジンなどを設定している人が多いでしょう。

サイトの最後のWebページから、そのユーザーのホームに移動させるといった使い方もできます。

DISPLAY





「ホームページ」に指定した
Webページが表示される

SOURCE

backhome.html

```
<html>
<title>go home</title>
<body bgcolor="#fff8dc">
<h3>ホームに戻る</h3>
<hr />
<form>
    <input type="button" value="Homeに戻る" onclick="home()" />
</form>
</body>
</html>
```

①: onclick="home()"

DETAILS

このスクリプトでは、 Homeに戻る ボタンをクリックすることにより、ホームに戻ります。このhomeというメソッドは、

```
window.home()
```

という形で扱います。windowオブジェクトは省略可能です。ここでは、①のように記述して、フォームボタンがクリックされるとonClickイベントハンドラによってhomeメソッドが実行されるようにしています。

6-5 WIN 4↑ N 4↑ MAC 4↑ N 4↑

フレームの内容を変更する

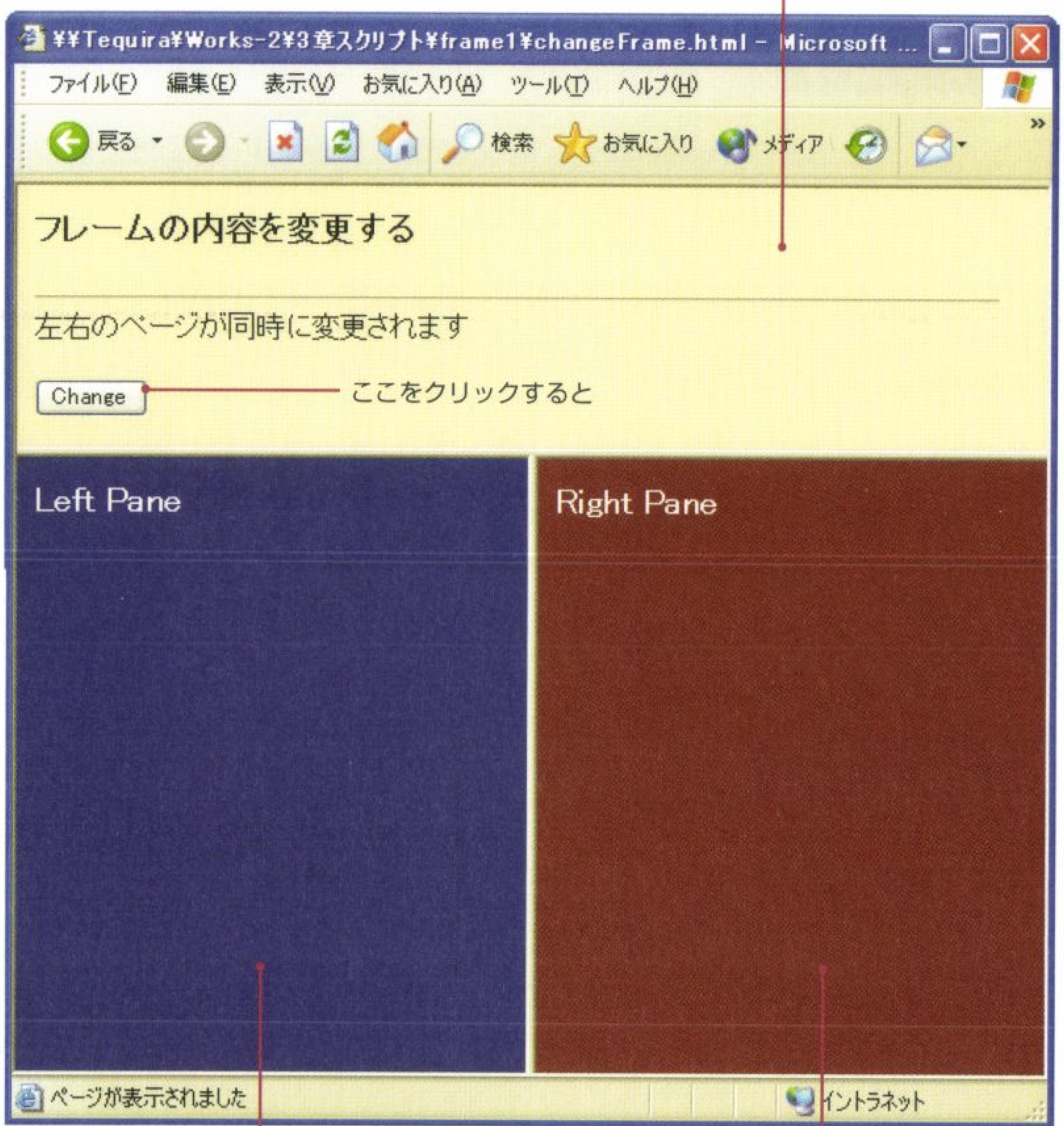
```
parent.frameオブジェクト名.location.href = "URL";
```

フレームでウィンドウを分割した場合、それぞれのフレームページを変更する作業はかなりの頻度で行われます。このサンプルでは、別フレームに表示されるページをJavaScriptで変更しています。フレームページの変更にスクリプトを使用すると、タイマーと組み合わせて一定時間後に自動的に他のフレームのページを切り替えたり、複数のフレームページを1回の操作で変更させることなども可能です。

PAGE

DISPLAY

「main」フレーム
(frame.html)



「left」フレーム
(left.html)

「right」フレーム
(right.html)



それぞれのフレームに別
のページが読み込まれる

SOURCE changeFrame.html

```
<html>
<frameset rows="30%,*">
<frame src="frame.html" name="main">
   <frameset cols="50%,*">
      <frame src="left.html" name="left">
      <frame src="right.html" name="right">
   </frameset>
</frameset>
</html>
```

①

frame.html

```
<html>
<title>frame change</title>

<script language="javascript">
<!--
function changeContents(){
   parent.left.location.href="http://books.softbank.co.jp/";
   parent.right.location.href="http://www.neko-no-te.net/books/";
}
//-->
</script>

<body bgcolor="#fff8dc">
<h3>フレームの内容を変更する</h3>
<hr />
左右のページが同時に変更されます
<form>
   <input type="button" value="Change"
⇥      onclick= "changeContents()" />        ②
</form>
</body>
</html>
```

left.html

```
<html>
<title>left page</title>
<body bgcolor="#000080" text="#ffffff">
<h3>Left Pane</h3>
</body>
</html>
```

right.html

```
<html>
<title>right page</title>
<body bgcolor="#8b0000" text="#ffffff">
<h3>Right Pane</h3>
</body>
</html>
```

DETAILS

このサンプルは、フレームによってウィンドウが3つに区切られています。上に表示されているフレームページのフォームボタンをクリックすると、下にある2つのフレームページが同時に変更されます。

フレームを設定するchangeFrame.htmlでは、上のフレームを「main」、左のフレームを「left」、右のフレームを「right」という名前にしています(①の箇所)。それぞれのフレームに表示させるページを設定する際には、この名前を参照します。

「main」フレームにある[Change]ボタンをクリックすると、②の<input>タグのonclickイベントハンドラによって、changeContents関数が呼び出されます。

changeContents関数では、左右それぞれのフレームに表示させるWebページを変更します。他のフレームのWebページの内容を変更する場合には以下のようになります。

```
parent.frameオブジェクト名.location.href = "URL";
```

parentは現在のフレームの親フレームを表すプロパティです。この例のように「main」フレームのページに記述してあるスクリプトから、同じ階層にある「left」フレームや「right」フレームへのアクセスは以下のようになります。

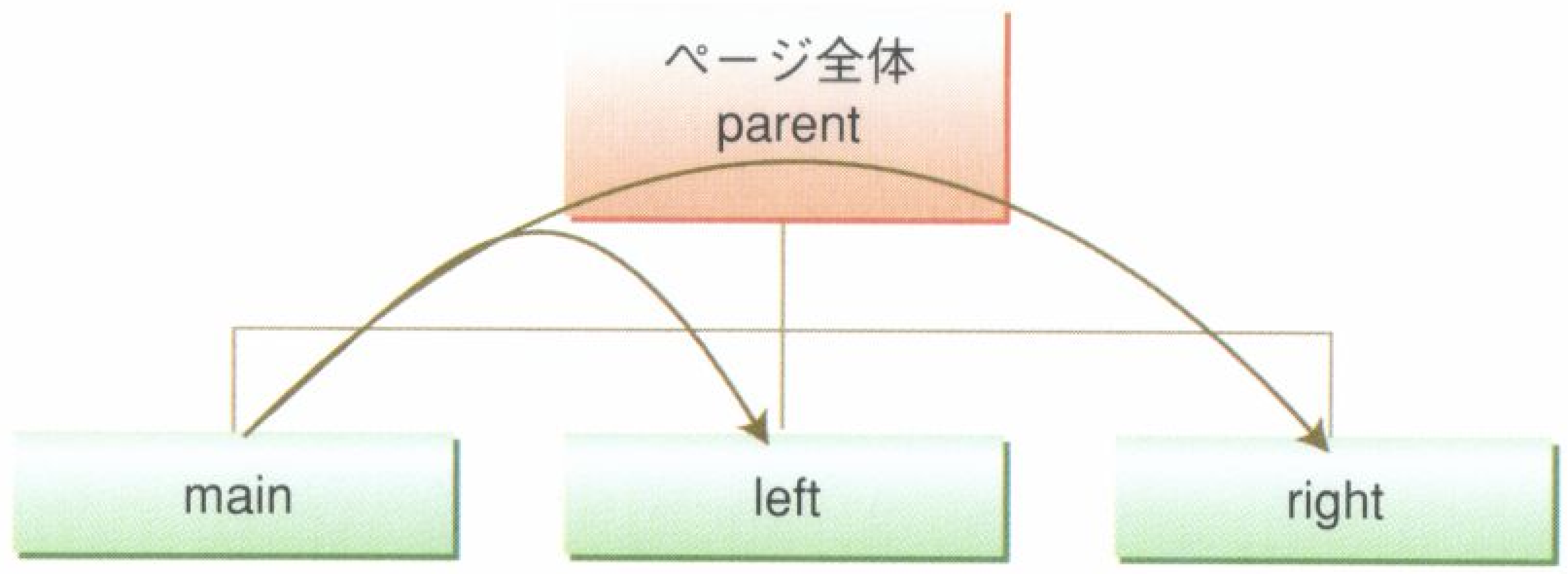
```
parent.left
parent.right
```

これはちょうどMS-DOSの相対パス指定と同じものだと考えればわかりやすいでしょう。たとえば同じ階層に「main」、「left」、「right」と3つのディレクトリがあったとして、「main」がカレントディレクトリのときに、「left」や「right」は次のように表現します。

```
..¥left
..¥right
```

この「..」にあたるのがparentプロパティです。

また、ルートディレクトリを表す「¥」にあたるtopプロパティや、カレントディレクトリを表す「.」にあたるselfプロパティなども用意されています。



実際のスクリプトでは、フレームのlocationオブジェクトのhrefプロパティにURLを与えています。

```
parent.left.location.href="http://books.softbank.co.jp/";
```

これにより、「left」フレームに指定したURLのWebページが読み込まれます。右側のフレームにも同様の設定を行います。その結果、ボタンをクリックしたときに、左右のフレームページが順番に変更されるのです。

なお、この例の場合、「left」フレームの親フレームはページ全体ですので、次のようにtopプロパティを使って記述しても、同じ動作をします。

```
top.left.location.href="http://books.softbank.co.jp/";
```

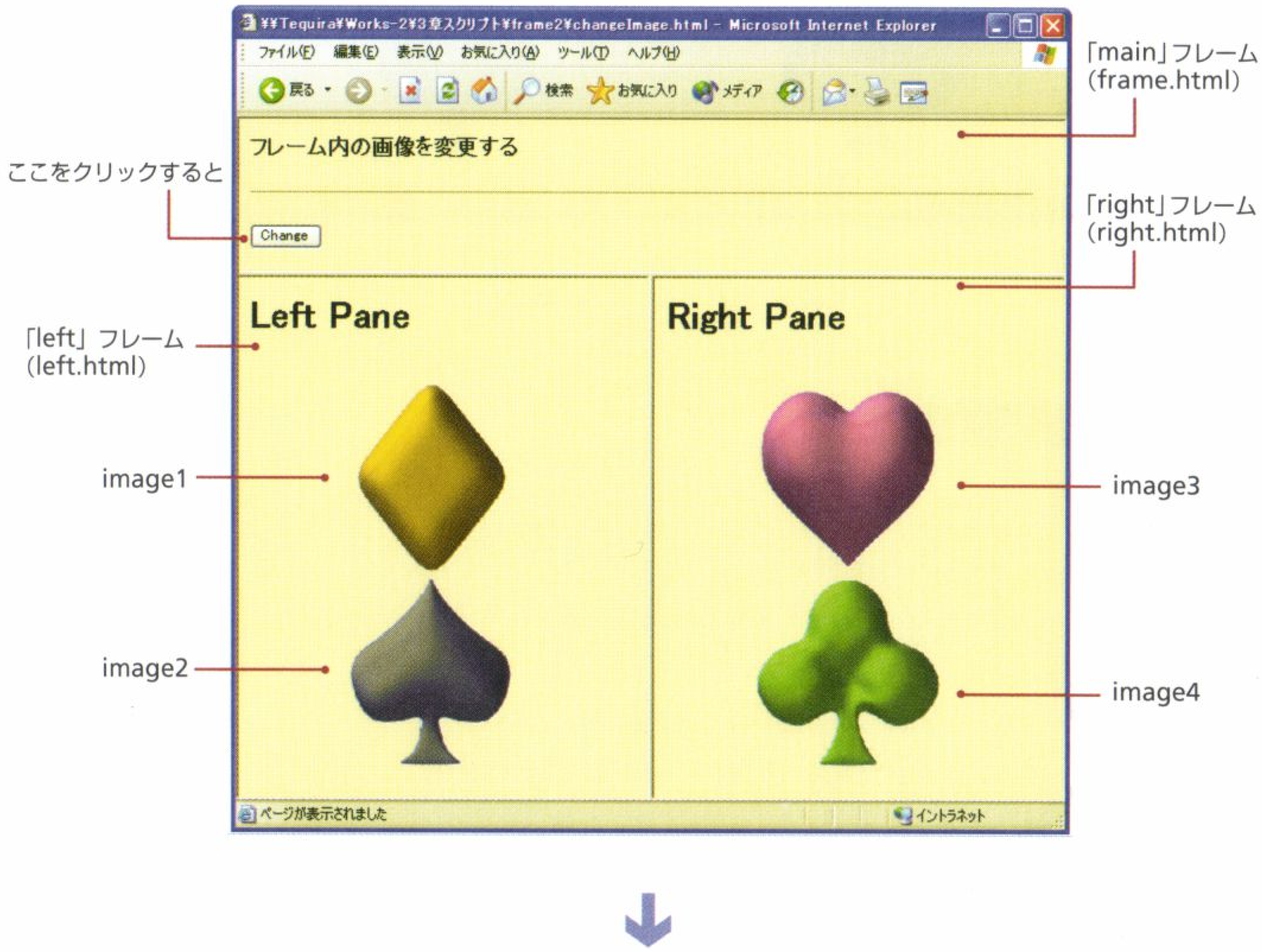
6-6 フレームのページの画像を変更する

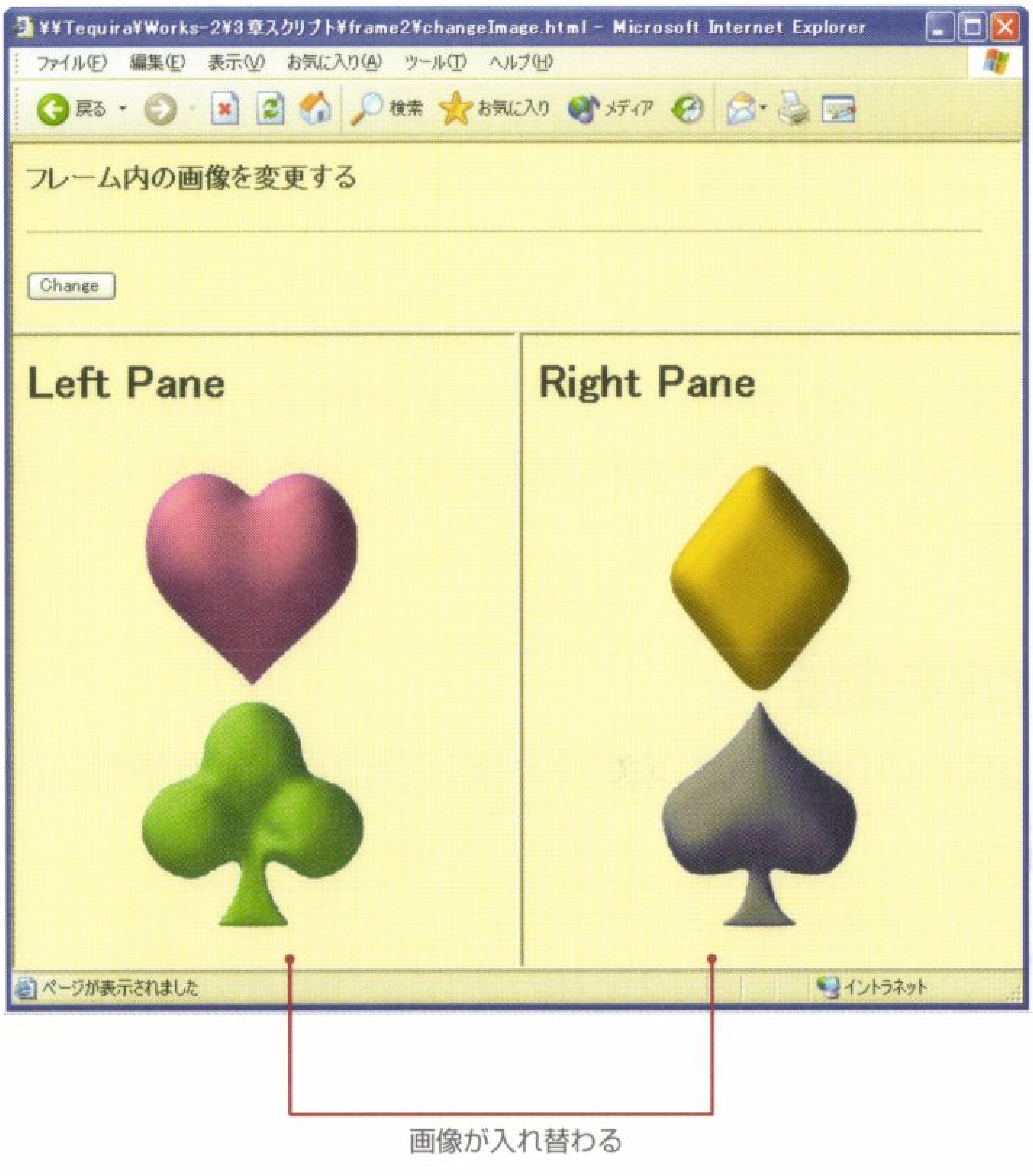
WIN / MAC

```
parent.frameオブジェクト名.document.image
オブジェクト名.src = "URL"
```

フレームを効果的に使用した、使い勝手のよいWebページをよく見かけます。ここでは、別フレームに表示されている画像を変更する方法を説明します。応用すれば、他のフレームページのさまざまなオブジェクトを参照または設定することも可能です。

DISPLAY





SOURCE

changeImage.html

```
<html>
<frameset rows="130,*">
<frame src="frame.html" name="main">
   <frameset cols="50%,*">
      <frame src="left.html" name="left">
      <frame src="right.html" name="right">
   </frameset>
</frameset>
</html>
```

frame.html

```
<html>
<title>frame image change</title>

<script language="javascript">
<!--
```

```
function changeImage(){
    parent.left.document.image1.src="images/img3.gif";
    parent.left.document.image2.src="images/img4.gif";        ④
    parent.right.document.image1.src="images/img1.gif";
    parent.right.document.image2.src="images/img2.gif";
}
//-->
</script>

<body bgcolor="#fff8dc">
<h3>フレーム内の画像を変更する</h3>
<hr />
<form>
    <input type="button" value="Change"
⇒       onclick= "changeImage()" />
</form>                                                      ③
</body>
</html>
```

left.html

```
<html>
<body bgcolor="#fff8dc">
<h1>Left Pane</h1>
<div align="center"><br />
    <img src="images/img1.gif" name="image1"
⇒       width="160" height="160" /> <br />                  ①
    <img src="images/img2.gif" name="image2"
⇒       width="160" height="160" /> </div>
</body>
</html>
```

right.html

```
<html>
<body bgcolor="#fff8dc">
<h1>Right Pane</h1>
<div align="center"><br />
```

```
    <img src="images/img3.gif" name="image1"
⇨       width="160" height="160" /> <br />
    <img src="images/img4.gif" name="image2"
⇨       width="160" height="160" /> </div>
</body>
</html>
```

②

DETAILS

このサンプルは、4つのファイルから構成されています。フレームを設定するファイルと、それぞれのフレームに表示される3つのページのファイルです。画面上は3つのフレームに区切られていて、上のフレームにあるボタンをクリックすると、それぞれのフレームページは更新されることなく、フレームページ内の画像のみが変更されます。

フレームを設定するchangeImage.htmlでは、上のフレームに「main」、左下のフレームに「left」、右下のフレームに「right」という名前を付けています。この名前は、それぞれのフレームを参照する際に使用されます。

「left」フレームに読み込まれたWebページのイメージ(Imageオブジェクト)には、それぞれ「image1」、「image2」という名前が付けられています(①の箇所)。「right」フレームに読み込まれるファイルのイメージにも同様に「image1」、「image2」という名前が付けられています(②の箇所)。

今回のサンプルで重要なのは、「main」フレームに表示されるファイルです。フォームボタンを配置し、クリックしたときにchangeImage関数を呼び出すようにonclickイベントハンドラで指定しています(③の箇所)。

④の部分で、changeImage関数が行う処理は、画像を取り換えるだけの簡単なものです。同じフレーム内のWebページの画像を変更する場合には、documentオブジェクト内にある画像オブジェクトのsrcプロパティに画像のURIを設定しますが、この例のように、他のフレームにある画像を設定するには、P.195でも紹介したparentプロパティを使用します。

```
parent.frameオブジェクト名.document.imageオブジェクト名.
⇨  src = "URL"
```

このサンプルでは、次のような記述で指定しています。

```
parent.left.document.image1.src = "img3.gif"
```

leftは左下のフレームに付けた名前です。そして、そのフレームのdocumentオブジェクト内にあるimage1というImageオブジェクトのsrcプロパティの値を「img3.gif」に変更します。その結果、左上の画像がそれまで右下にあったものに変更されるのです。他の画像についても同様に変更されます。なお、左右両方のフレームの画像オブジェクトに、同じimage1という名前が設定されていますが、フレームの名前によって識別されるので、別のフレームに同じ名前のオブジェクトがあっても正しく参照できます。

COLUMN

コードの字下げ

JavaScriptはスクリプト言語とはいえ、結構長いコードになってしまうことがあります。そんなときには少しでも見やすくするために工夫が必要です。

たとえば、以下のようなコードはあまり見やすくありません。

```
for( i=0; i<=7 ; i++ ){
document.write("<font size=" + i + ">hello</font><br />");
j=j+1;
}
```

このようなコードを見やすくするには字下げを行います。

```
for( i=0; i<=7 ; i++ ){
    document.write("<font size=" + i + ">hello</font><br />");
    j=j+1;
}
```

for文によりループ内で行われる処理の前に空白(タブ)を入れて、ループがどこからどこまでかをわかりやすくしています。同様にif文やwhile文なども字下げをして記述するべきです。{} で囲まれている場合には、その中の記述は外側よりも字下げを行うようにします。通常タブにより字下げをするのが、プログラミング上効率的です。

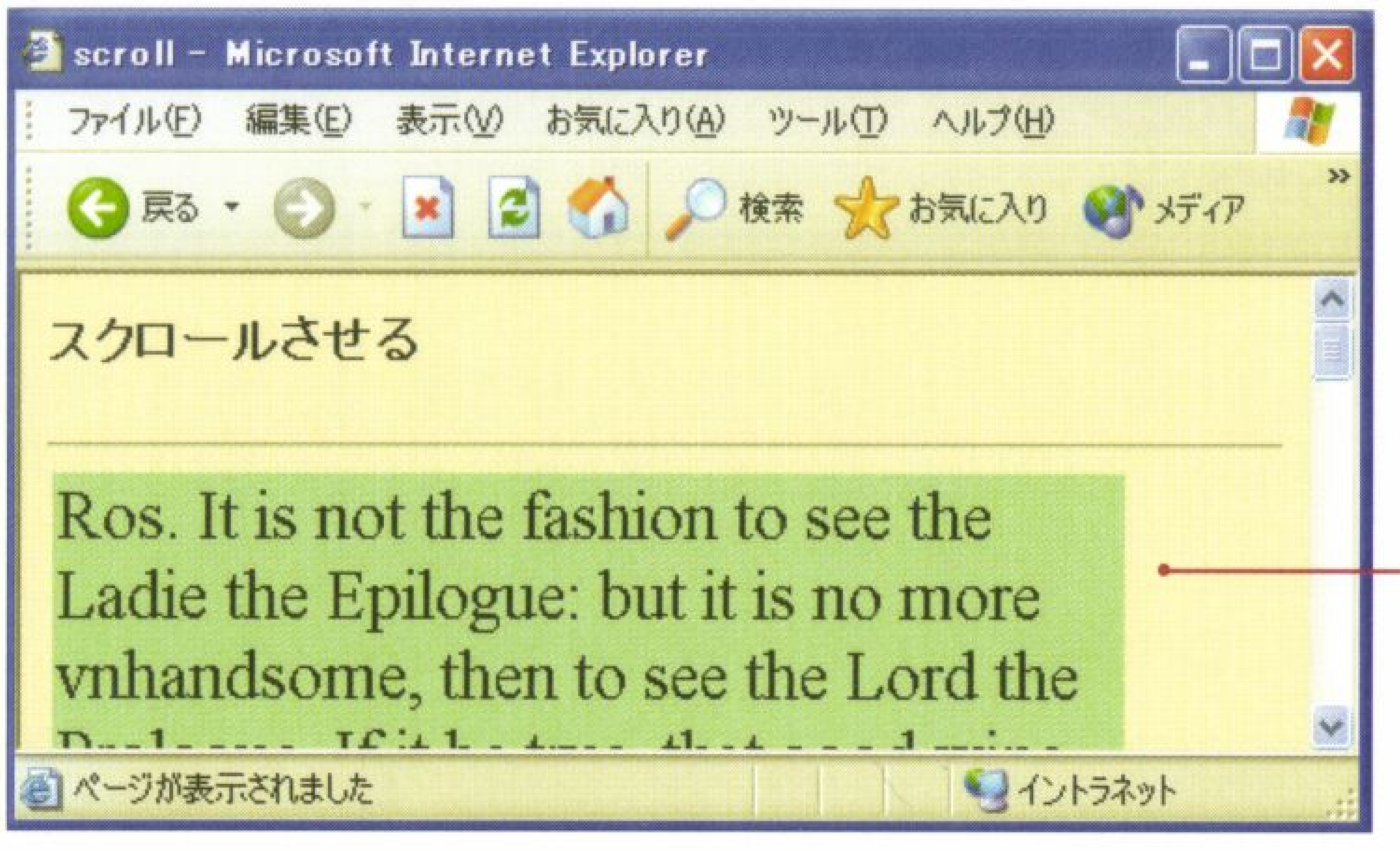
PAGE

ページの自動スクロール

```
windowオブジェクト名.scroll(X座標, Y座標);
setTimeout("命令", 間隔);
```

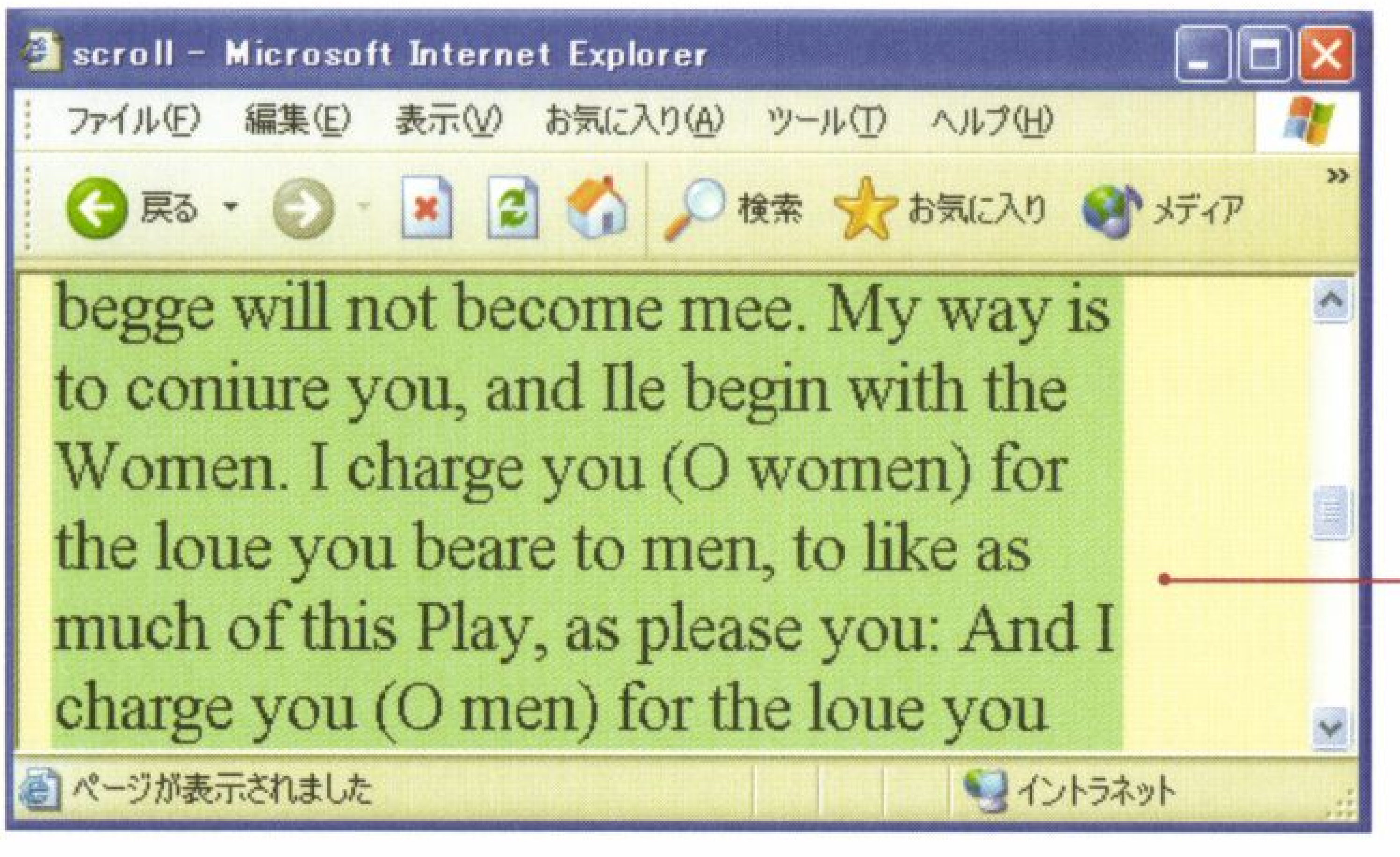
映画のスタッフロールなどでよく見られる、画面がスクロールするサンプルです。うまく使うとよい効果を出すことができるでしょう。

D I S P L A Y



ページが読み込まれるのと同時に、
自動的にスクロールが始まる

↓



どんどん下へスクロールする

↓



SOURCE scroll1.html

```
<html>
<title>scroll</title>

<script language="javascript">
<!--
var stop_y = 1000;
var dy = 0;                                   ②
var step = 5;
var sec = 10;

function windowScroll(){
   if (dy < stop_y) {
      dy += step;                              ①
      window.scroll(0, dy);
      setTimeout("windowScroll()", sec);
   }
}
// -->
</script>

<body onload="windowScroll()" bgcolor="#fff8dc">
<h3>スクロールさせる</h3>
<hr />
```

```
<table border="0" height="1000" width="400">
<tr>
   <td bgcolor="#99ffcc" valign="top">
<div style="font-size: 25px; font-family: Times">
Ros. It is not the fashion to see the Ladie the Epilogue:

(略)

 offer, when I make curt'sie, bid me farewell.
</div>
   </td>
</tr>
</table>
</body>
</html>
```


DETAILS

このサンプルではWebページがロードされるとウィンドウ内がスクロールするようになっています。スクロールを実際に行うのは、①の部分のwindowScroll関数です。この関数内ではwindowオブジェクトのscrollメソッドを用いてスクロールを実現しています。scrollメソッドは、以下のような書式になります。

```
windowオブジェクト名.scroll(X座標, Y座標);
```

scrollメソッドで用いるX座標、Y座標は絶対座標です。この座標はWebページの左上を(0,0)として右方向をX方向、下方向をY方向としているものです。



windowScroll関数で使用する変数は、②の部分に設定されています。Webページは、変数dyに設定されている値の位置に移動(スクロール)します。この変数dyは初期値が0ですが、処理の途中で変数stepに設定されている値だけ足されます。しかし、scrollメソッドを1回だけ実行しても、変数stepの値しかスクロールしません。そこで用いるのが、setTimeoutメソッドです。このメソッドは一定時間ごとに命令を呼び出すものです。setTimeoutの書式は以下のようになります。

```
setTimeout("命令", 間隔);
```

このメソッドの間隔の単位はミリ秒(1/1000秒)です。1000で1秒となります。このサンプルでは変数secに10を設定して、10ミリ秒ごとにwindowScroll関数を呼び出しています。

さらにこのサンプルではどこまでスクロールさせるかを指定するために、変数stop_yを用意しています。この変数stop_yをwindowScroll関数の中で最初に比較することにより、スクロールの量をコントロールしています。変数dyが変数stop_yを超えたらwindowScroll関数は実行されなくなります。

PAGE

TIPS

自動スクロールをWebブラウザ内で実際に行うと、ユーザーがコントロールできないため使いづらい場合があります。そのため、よく用いるのはフローティングフレーム内で動作させるというものです。フローティングフレームはWebページ内の任意の場所にフレームを作れるというものです。フローティングフレームは<iframe>タグを用いて記述します。

```
<iframe src="URI"></iframe>
```

今回のスクロールするファイルをsrc属性に指定することにより、フローティングフレーム内でスクロールさせることができます。また、その他の属性として、幅(width)、高さ(height)、枠線の有無(frameborder)、スクロール(scrolling) なども指定することが可能です。

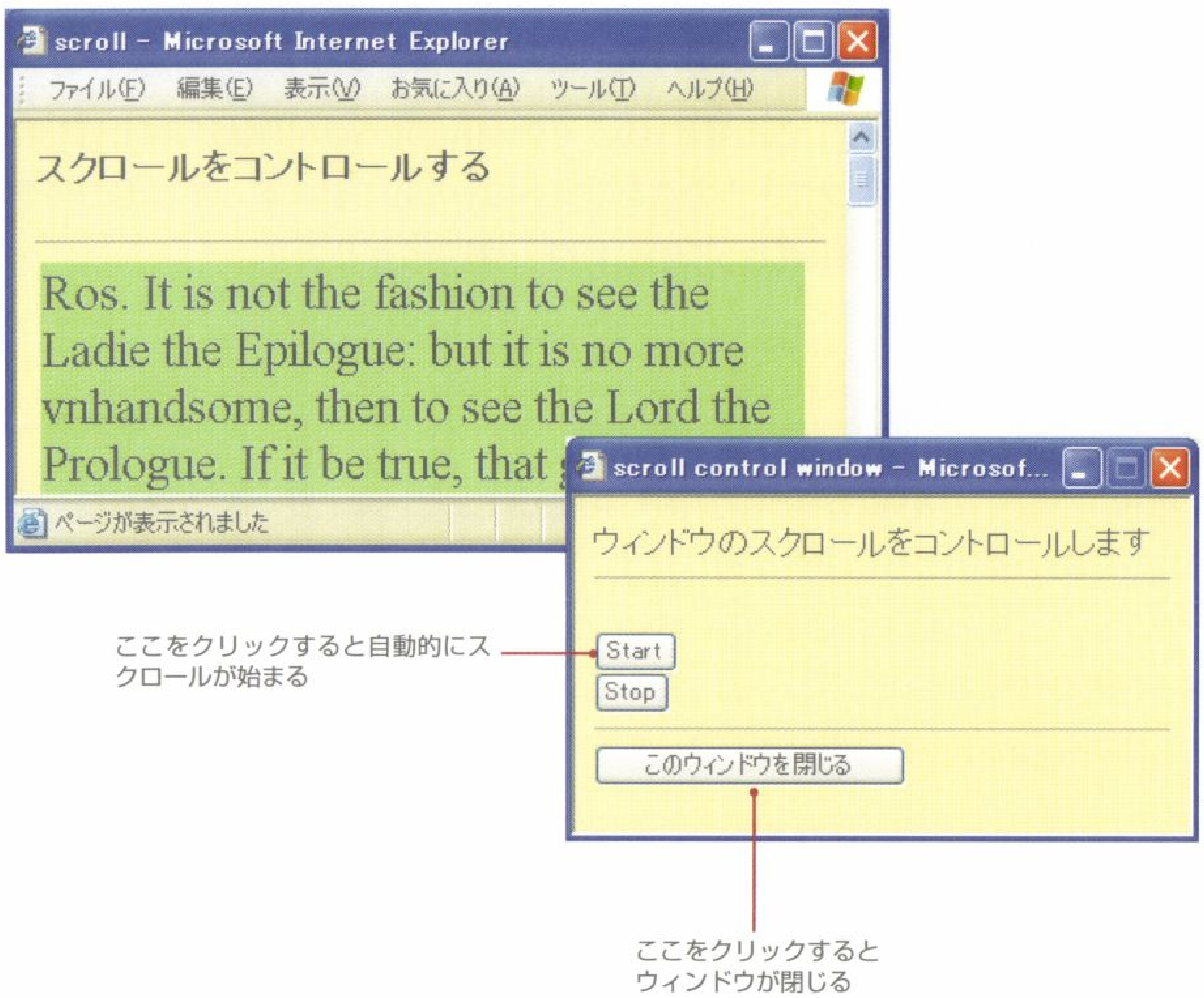
6-8 スクロールをコントロールする

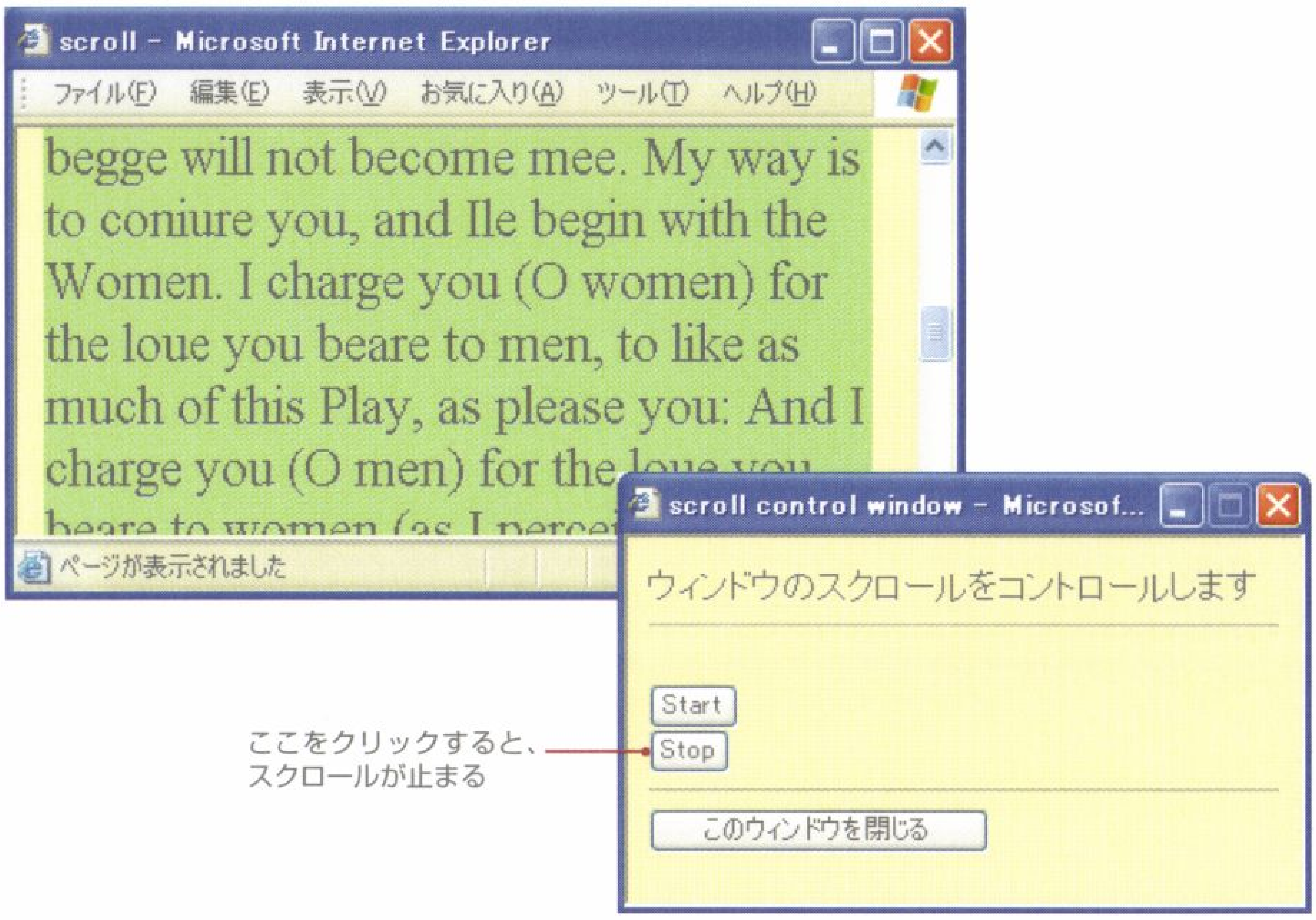
WIN MAC

```
windowオブジェクト名.opener.scroll(X座標, Y座標);
タイマーID = setTimeout("命令", 間隔);
clearTimeout(タイマーID);
```

スクロールは結構面白いことができるのですが、このスクロールをユーザーがコントロールできるようにしたいものです。スクロールさせることもよいのですが、スクロールを止められないと困るという内容もあります。そんな場合には、スクロールのコントロール機能を付けると有効です。

DISPLAY





S O U R C E scroll2.html

```
<html>
<title>scroll</title>

<script language="javascript">
<!--
function openWin(winname){
   window.open("scrollControl.html",winname,"width=360,
      height=200,toolbar=no,location=no,status=no,menubar=no,
      scrollbars=no");
}
// -->
</script>

<body bgcolor="#fff8dc" onload="openWin('contorlwin');">
<h3>スクロールをコントロールする</h3>
<hr />
<table border="0" height="1000" width="400">
<tr>
   <td bgcolor="#99ffcc" valign="top">
```

```
<div style="font-size: 25px; font-family: Times">
Ros. It is not the fashion to see the Ladie the Epilogue:

(略)

offer, when I make curt'sie, bid me farewell.
</div>
   </td>
</tr>
</table>
</body>
</html>
```

scrollControl.html

```
<html>
<title>scroll control window</title>
<script language="javascript">
<!--
var stop_y = 1000;
var dy = 0;
var step = 5;
var sec = 10;
var bstatus = 'true';
var timerID;

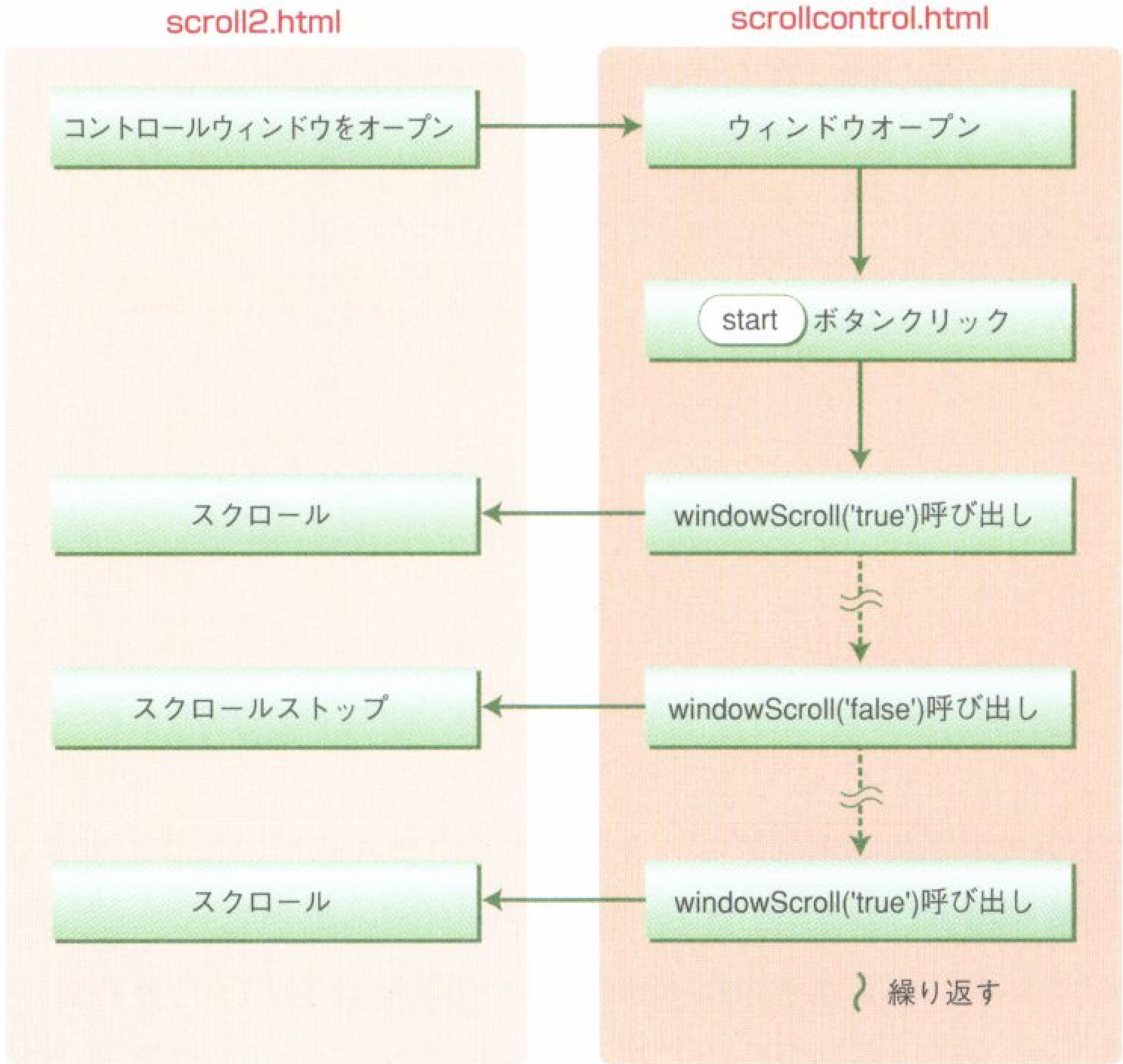
function windowScroll(bstatus) {
   if( bstatus=='false'){                                        ②
      clearTimeout(TimerID);
   }
   else if (dy < stop_y) {
      dy += step;
      window.opener.scroll(0, dy);                               ①
      TimerID = setTimeout("windowScroll(bstatus)", sec);
   }
}
//-->
</script>

<body bgcolor="#fff8dc">
```

```
ウィンドウのスクロールをコントロールします
<hr />
<form>
  <input type="button" value="Start" onclick="windowScroll
⇒     ('true')" /><br />
  <input type="button" value="Stop" onclick="windowScroll
⇒     ('false')" />
<hr />
  <input type="button" value="このウィンドウを閉じる"
⇒     onclick="window.close()" />
</form>
</body>
</html>
```

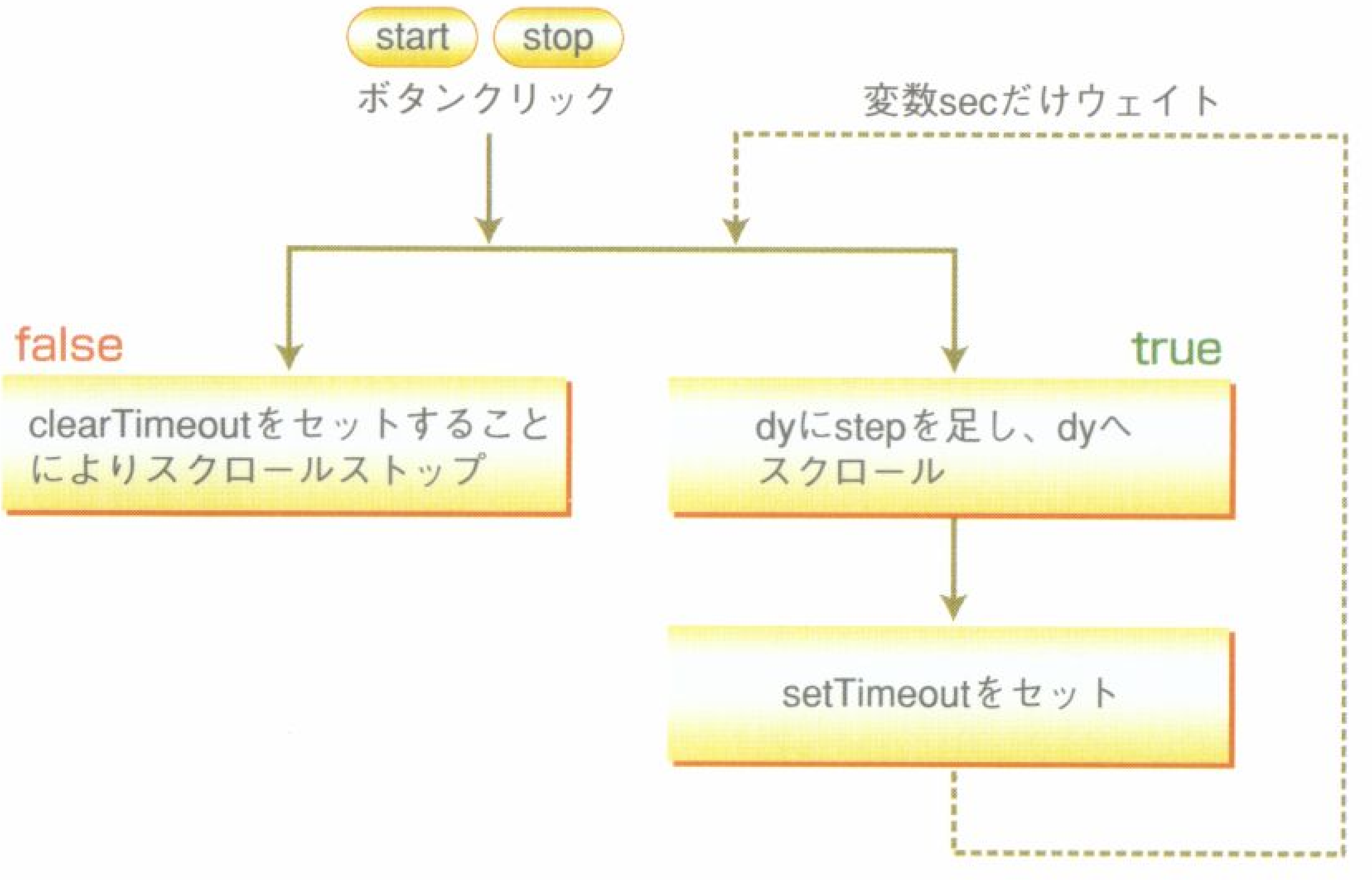

DETAILS

このサンプルは2つのファイルで構成されています。1つはスクロールされる内容が記述してあるHTMLファイル、もう1つはスクロールをコントロールするHTMLファイルです。



scroll2.htmlの中で、ロード時にスクロールをコントロールするscrollControl.htmlを開いています。そのため、スクロールするscroll2.htmlが親ウィンドウとなります。scroll2.htmlには、scrollControl.htmlを開くという機能以外は記述されていません。scrollControl.htmlでは[Start]と[Stop]のボタンを使ってスクロールをコントロールすることができます。スクロールが終了するまではスクロールをコントロールすることができます。

では、次にスクロールをコントロールするscrollControl.htmlの記述について解説していきます。このWebページには3つのフォームボタンが配置されています。このうちの[Start]ボタンと[Stop]ボタンのonclickイベントハンドラによりwindowScroll関数の呼び出しを行い、scroll2.htmlのスクロールをコントロールしています。このwindowScroll関数はP.202のサンプルで解説したものを改良しています。この改良の中で重要なのは、関数の呼び出しに引数を付け、これによりスクロールの状態をコントロールできるようにしている部分です。引数としては「true」、「false」の2種類があります。trueのときにはスクロールし、falseのときにはスクロールが止まるようにしています。



windowScroll関数の内部では、上図のような動作を行います。まず引数として渡された値を格納した変数bstatusがtrueとfalseのどちらなのかを判定し、trueの場合であれば、変数stepに格納された値を変数dyに足し合わせ、変数dyの位置までscrollメソッドを用いてスクロールさせます(①の箇所)。このとき、

どこをスクロールさせるかという指定を見ると、次のようになっています。

```
window.opener.scroll(0, dy);
```

この記述のopenerというのは、「このウィンドウを開いたウィンドウ」を表すキーワードです。これで、scrollControl.htmlを開いたscroll2.htmlを表すことができます。

その後に、setTimeoutメソッドを用いて変数secに格納されている値の分だけ時間が経過したら、windowScroll関数を呼び出すようにしています。なおsetTimeoutメソッドでは、変数TimerIDというものを設定しています。

```
タイマーID = setTimeout("命令", 間隔);
```

これはWebページ内で、setTimeoutやsetIntervalを複数用いたときに区別を付けるためのものです。こういった働きをする変数を一般的にはタイマーIDと呼んでいます。

windowScroll関数が呼び出されたときの引数がfalseだった場合には、clearTimeoutメソッドが呼び出されています(②の箇所)。これはsetTimeoutメソッドで設定されている関数の呼び出しをクリアするものです。ここではsetTimeoutメソッドで設定されているタイマーIDをパラメータとして取ります。

```
clearTimeout(タイマーID);
```

これにより、setTimeoutメソッドで設定されているwindowScroll関数の呼び出しをクリアすることになります。

6-9

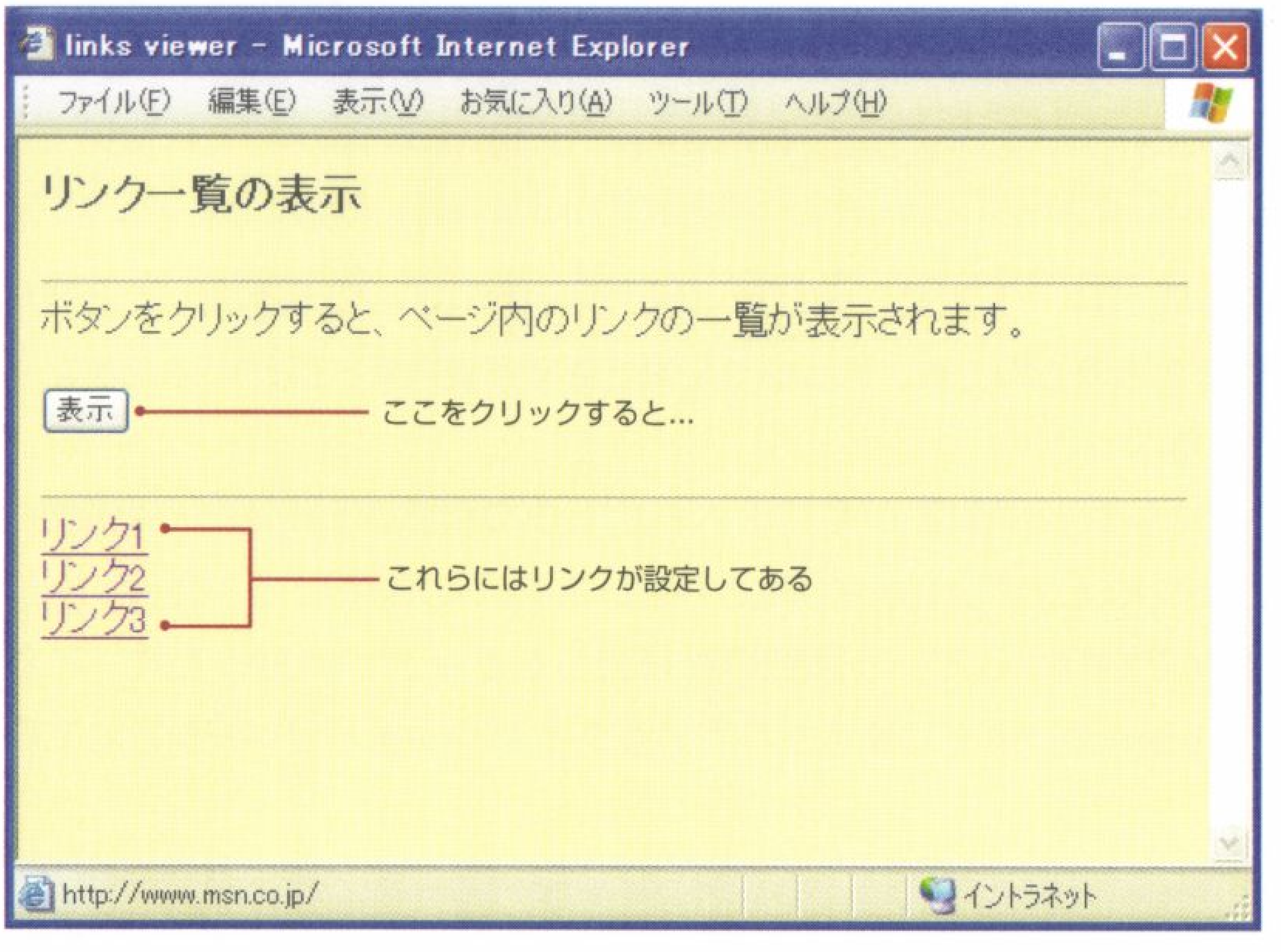
WIN 4↑ N 4↑ MAC 4↑ N 4↑

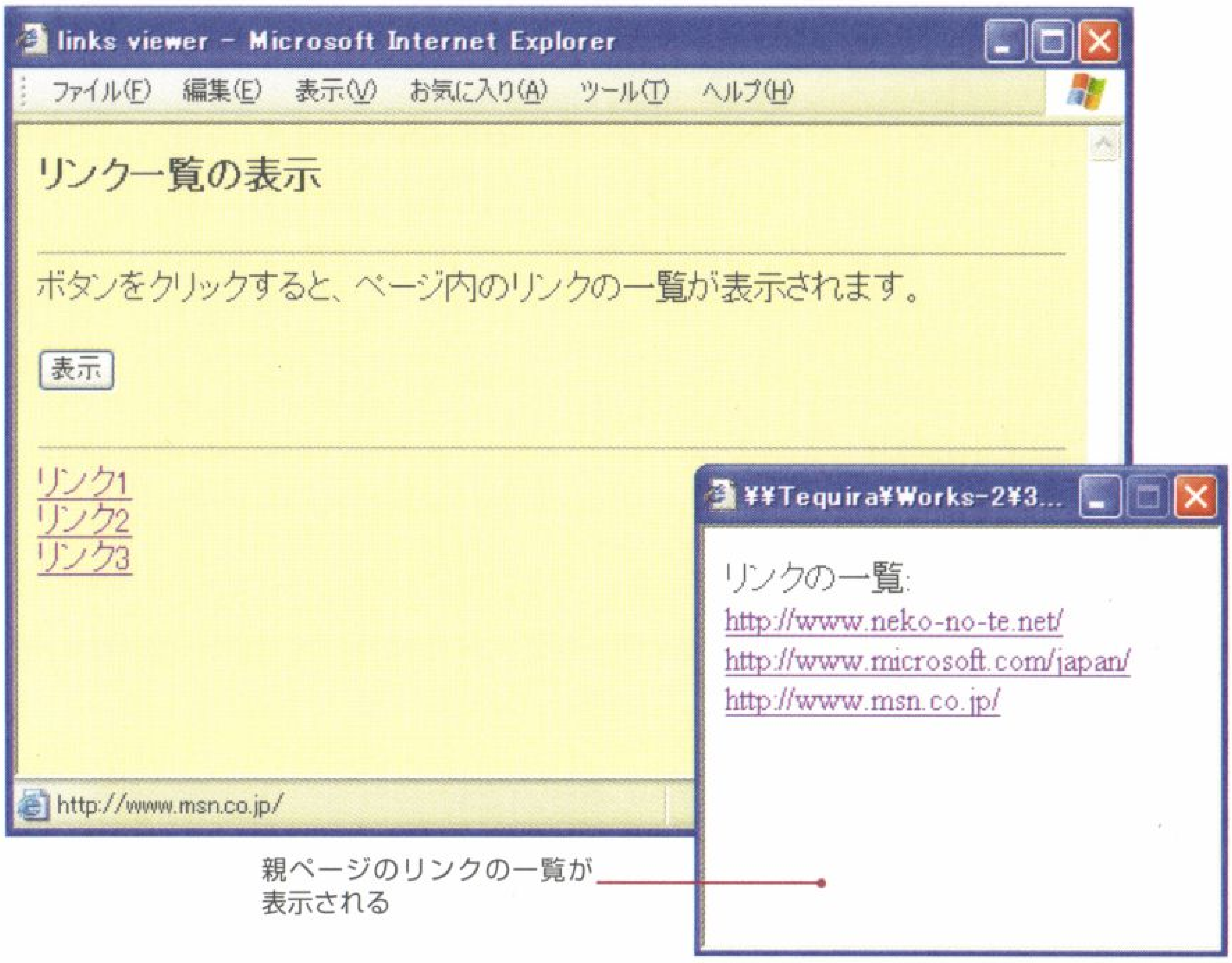
Webページ上のリンクの一覧を表示する

```
document.links.length
document.links[添え字]
```

JavaScriptを使用すると、Webページ上にどれだけの数のリンクがあるかを確認できます。また、どこへのリンクがあるのかを参照することも可能です。

D I S P L A Y





SOURCE

linkcheck.html

```
<html>
<title>links viewer</title>
<script language="javascript">
<!--
   function showLinks(){
      var msg = "リンクの一覧:<br />";
      for(i=0; i<document.links.length; i++){
         msg = msg + "<a href=" + document.links[i] +
            ">" + document.links[i] + "</a><br />";
      }
      var win1 = window.open('','newwin1', 'toolbar=no,
         location=no,status=no,menubar=no, width=300,
         height=300');
      win1.document.write(msg);
      win1.document.close();
   }
//-->
</script>
<body bgcolor="#fff8dc">
```

```
<h3>リンク一覧の表示</h3>
<hr />
ボタンをクリックすると、ページ内のリンクの一覧が表示されます。<br />
<form>
   <input type="button" value="表示" onclick="showLinks()" />
</form>                                                    ①
<hr />
<a href="http://www.neko-no-te.net/">リンク1</a><br />
<a href="http://www.microsoft.com/japan/">リンク2</a><br />
<a href="http://www.msn.co.jp">リンク3</a>
</body>
</html>
```

PAGE

DETAILS

このサンプルでは、Webページ内に記述されたリンクの一覧を別ウィンドウに表示するものです。

フォームボタンの下に「リンク1」から「リンク3」までのリンクが用意されています。それぞれ異なるURLへのリンクです。「表示」と書かれたボタンをクリックすると、①の部分のonclickイベントハンドラによってshowLinks関数が呼び出されます。

showLinks関数では、最初に変数msgに「リンクの一覧 : ￥n」という文字列が代入されます。「￥n」は改行コードです。

次に、for文を使用したループ処理を行っています（②の箇所）。ループの条件は以下のようになっています。

```
for(i=0;i<document.links.length;i++)
```

カウンタ「i」の初期値は0です。"document.links.length"はWebページ内のリンクの総数を返すので、このループはリンクの数だけ繰り返して処理が行われることになります（for文についてはP.44参照）。

Webページ内のURLは、documentオブジェクトのlinksコレクションに格納されています。コレクションとはオブジェクトの集まりを意味し、linksコレクションには、Webページ上のLinkオブジェクトが上から順番に格納されています。Linkオブジェクトに格納されたリンクへのアクセスは添え字を用いて行います。書式は次のようになります。

```
document.links[添え字]
```

添え字は0から開始されます。そのため、このサンプルにはlinksコレクションに、以下のようにリンクが格納されています。

●links[0]……http://www.neko-no-te.net/
●links[1]……http://www.microsoft.com/japan/
●links[2]……http://www.msn.co.jp/

サンプルでは、リンクへのアクセスを行いながら変数msgに文字列を追加していきます。

for文によって、3回ループ処理が行われ、変数msgの中に表示したいリンクがすべて格納されます。

この後、スクリプトでは新しいウィンドウを開くための処理を行います(③の箇所)。新しいウィンドウを作成するにはwindowオブジェクトのopenメソッドを用います。ウィンドウを開く部分に関してはP.123を参照してください。

④の部分では開かれたウィンドウに対して、変数msgを記述し閉じます。これにより、新たに開かれたウィンドウ内に、リンクの一覧が表示されます。

COLUMN

Script Encoder

JavaScriptの欠点として、ユーザーがコードを見ることができるという点があります。たとえば、非常に高度なロジックをJavaScriptにより実装した場合でも、ユーザーから隠す手段がないということになります。これでは、商業利用をする場合に困ってしまうことがあります。これに対処するためにMicrosoft社はスクリプトをエンコードできる機能を提供しています。

●ダウンロード元

URL▶ http://www.microsoft.com/japan/developer/scripting/

上記のサイトからスクリプトエンコーダーを取得してください。現在(2001年11月)のバージョンは1.0となっています。

スクリプトエンコーダーは、コマンドラインから使うツールです。コマンドプロンプトからインストールされたフォルダに移動、またはフルパスでプログラムを起動します。

スクリプトエンコーダーのプログラムの実行は以下のように入力します。

```
> screnc [/?] [/s] [/f] <source> <destination>

  /? ヘルプ
  /s サイレントモード（ディスプレイに何も表示されない）
  /f 上書きが発生しても強制的に置き換える
  <source> 変換元ファイル名
  <destination> 変換後ファイル名
```

たとえば、printChar.htmlをエンコードして、printCharEnc.htmlを作る場合には以下のように入力します。

```
> screnc printChar.html printCharEnc.html
```

なお、スクリプトエンコーダーでエンコードを行うと、IE5以上でしか動作しなくなります。

JavaScript
Handbook
3rd edition

7-1 WIN MAC

ボタンクリックで画像だけを切り替える

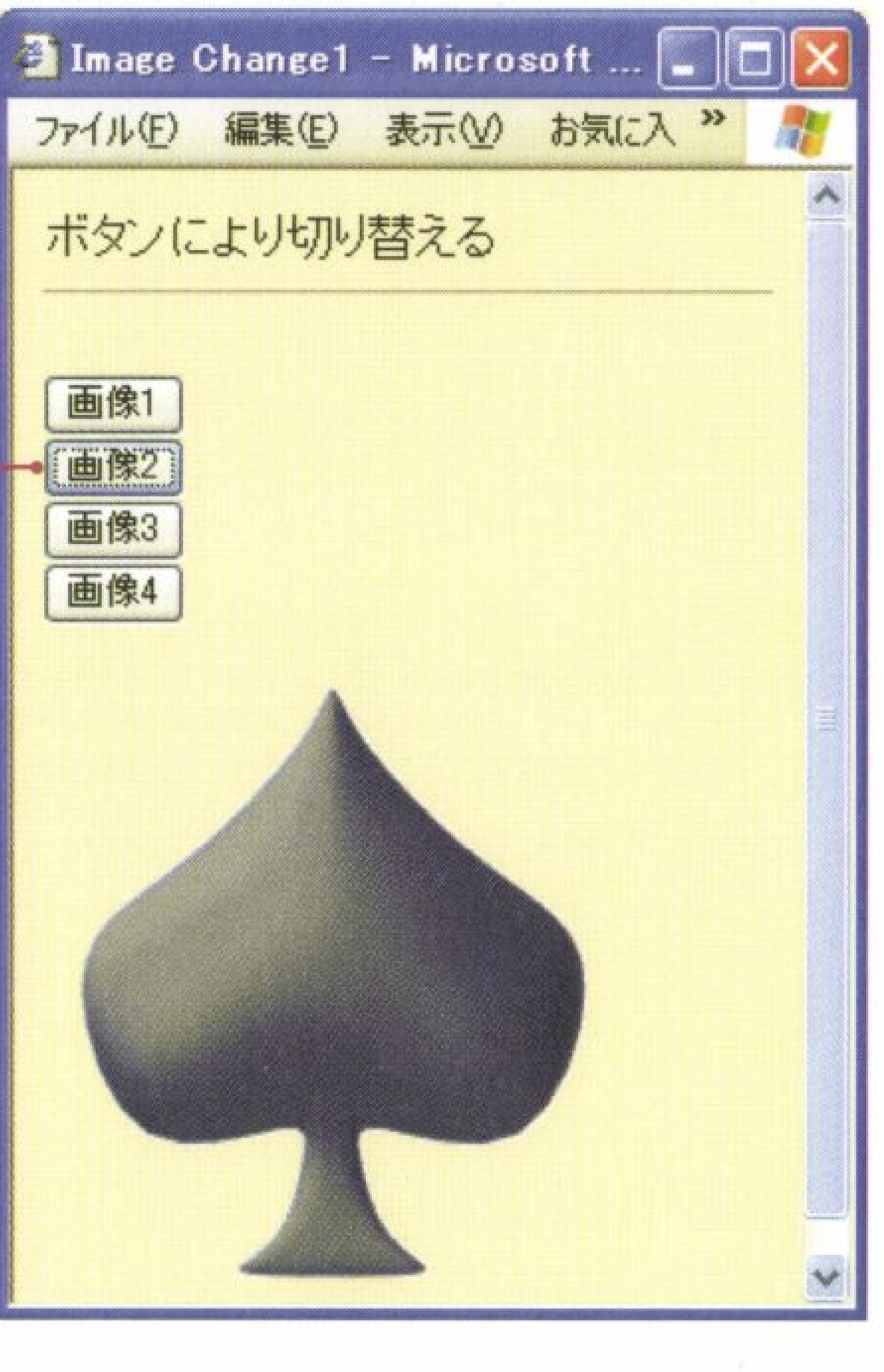
imageオブジェクト名 src="画像ファイル名

JavaScriptを使用すると、Webページ読み込み後に画像だけを切り替えることができます。いちいちWebページ全体を読み込まずに画像だけを変更できるので効率もよく、何よりも見た目の演出として効果的です。

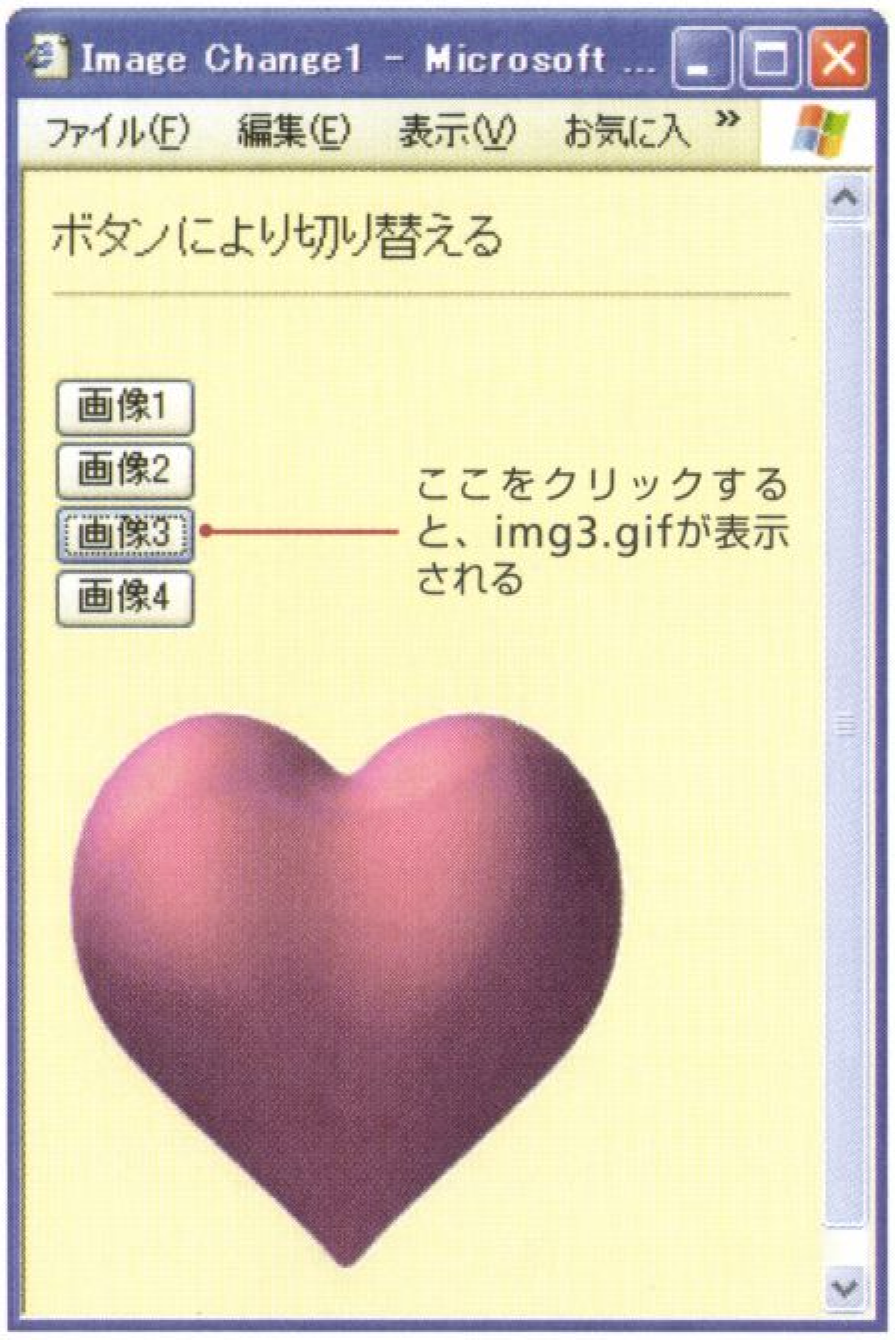
DISPLAY

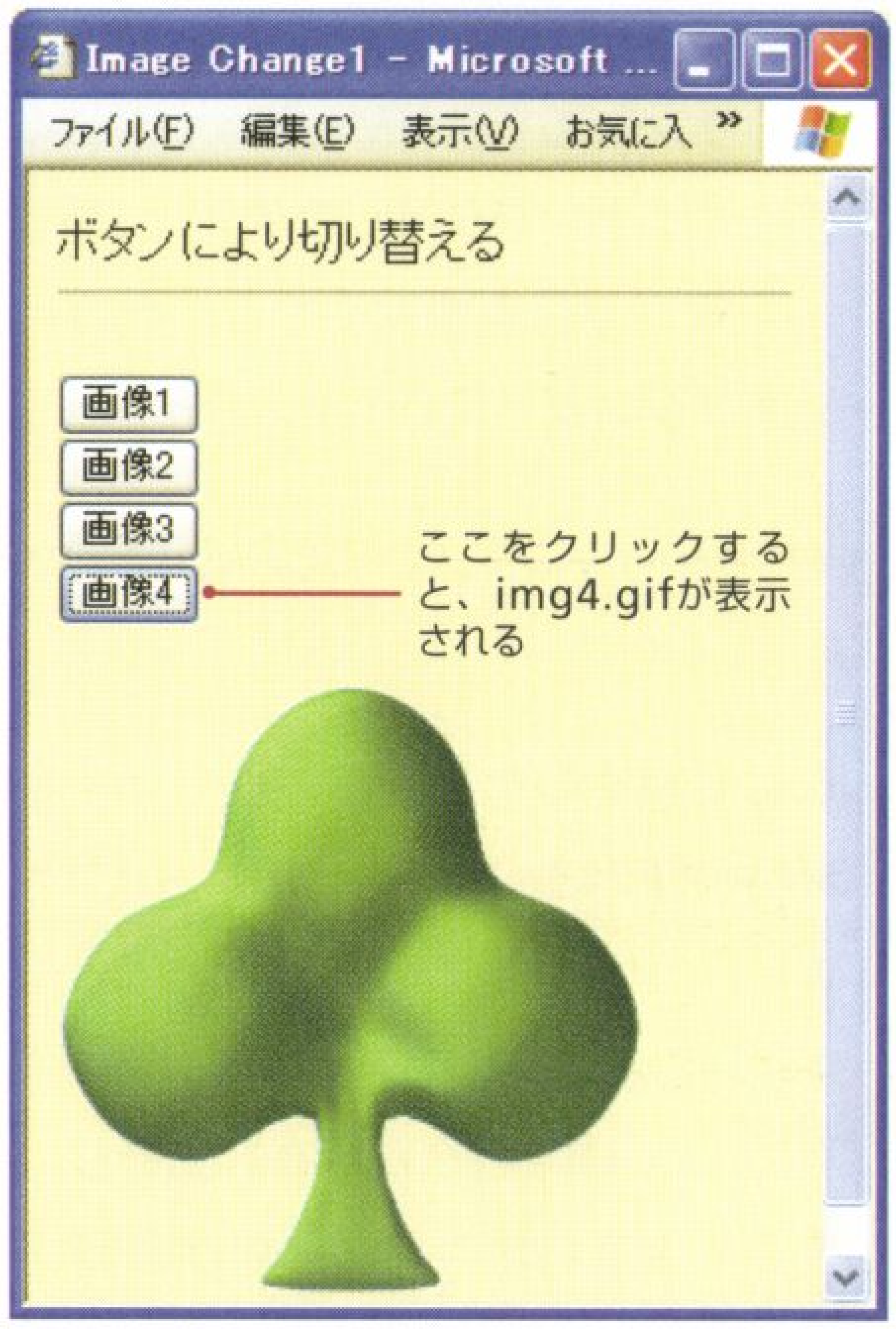


ここをクリックすると、下にimg1.gifが表示される



ここをクリックすると、img2.gifが表示される





SOURCE　　Imagechange1.html

```
<html>
<title>Image Change1</title>
<body bgcolor="#fff8dc">
ボタンにより切り替える<br />
<hr />
<form>
   <input type="button" value="画像1" onclick="myImg.src=
⇒      'images/img1.gif'" /><br />
   <input type="button" value="画像2" onclick="myImg.src=
⇒      'images/img2.gif'" /><br />                    ②
   <input type="button" value="画像3" onclick="myImg.src=
⇒      'images/img3.gif'" /><br />
   <input type="button" value="画像4" onclick="myImg.src=
⇒      'images/img4.gif'" /><br />
</form>
<br />
<img src="images/img1.gif" name="myImg" />            ①
</body>
</html>
```

DETAILS

このサンプルでは、Webページに4つのフォームボタンが配置されており、それぞれのボタンをクリックすると、4枚の画像が切り替わります。

①の部分の画像を表示するための<img>タグでは、name属性を用いてイメージオブジェクトに「myImg」という名前を付けています。

```
<img src="images/img1.gif" name="myImg" />
```

ここで指定した名前は、このImageオブジェクトを参照したり属性を設定する場合に使用されます。また、src属性の指定によって、Webページ読み込み時には、「img1.gif」が読み込まれて表示されます。

4つのフォームボタンには、value属性によって、それぞれ「画像1」、「画像2」、「画像3」、「画像4」と表示されており、onclickイベントハンドラによってボタンがクリックされたときの動作が指定されています。たとえば、「画像2」がクリックされると、onclickイベントハンドラに記述されている部分が実行されます(②の箇所)。

```
onclick="myImg.src='images/img2.gif'"
```

「myImg」は、先ほど指定したImageオブジェクトの名前です。これによってmyImgのプロパティの1つであるsrcプロパティが設定されます。ここではmyImgのsrcプロパティが「img2.gif」に設定され、画像が切り替わります。

TIPS

HTMLの要素(タグ)の属性の値は、ダブルクォーテーション(")かシングルクォーテーション(')で括るのがルールです。たいていのWebブラウザでは括らなくてもHTMLを解釈してくれますが、注意はしておいてください。

JavaScriptでは、文字列を使用する際にダブルクォーテーションもしくはシングルクォーテーションで括ります。ダブルクォーテーション自体を文字列として使いたい場合は、次のようにシングルクォーテーションで括ります。

```
onclick='alert("Hello JavaScript!")'
```

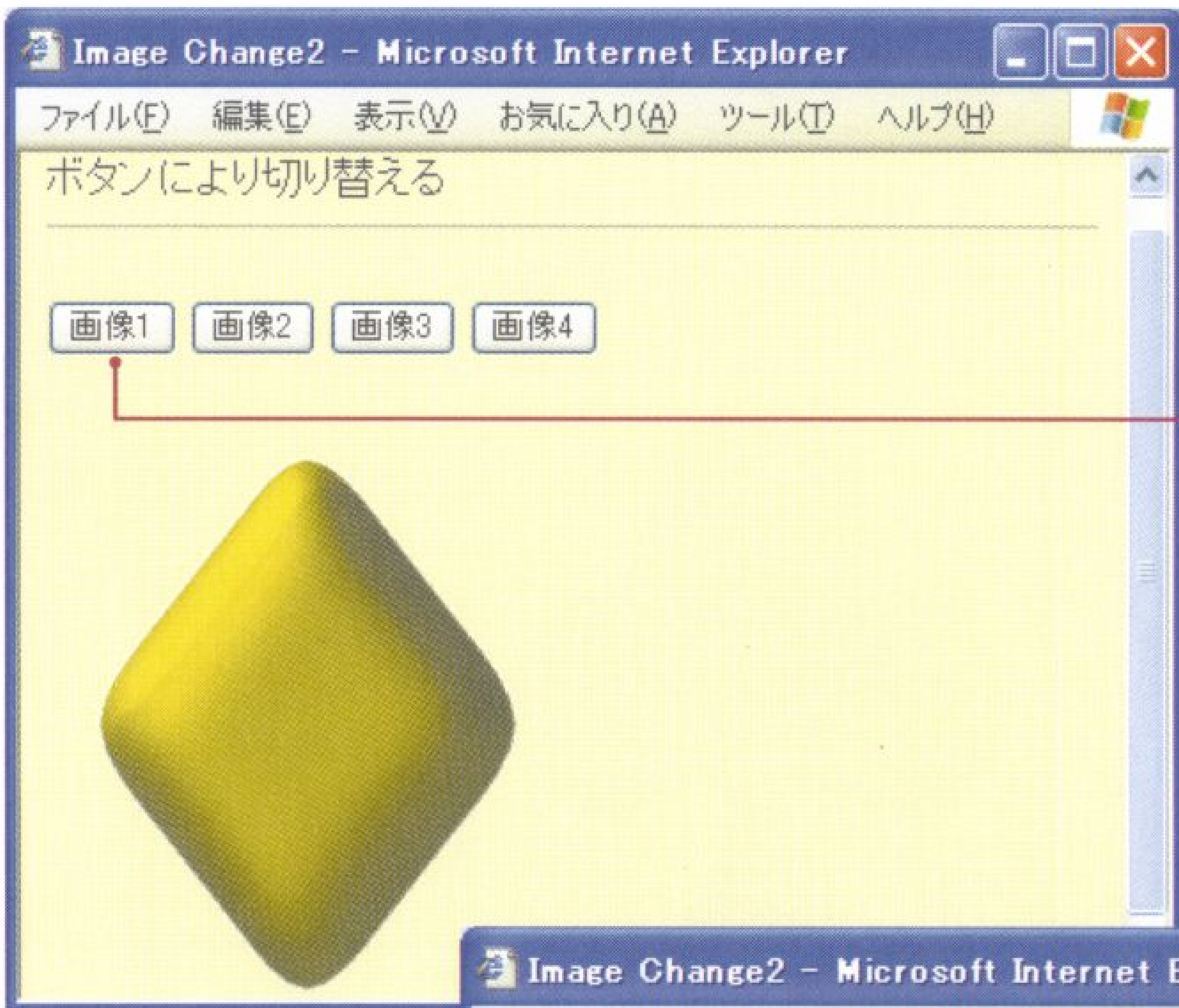
シングルクォーテーションを文字列として使う場合はこの逆になります。

ボタンクリックで画像を切り替える(拡張版)

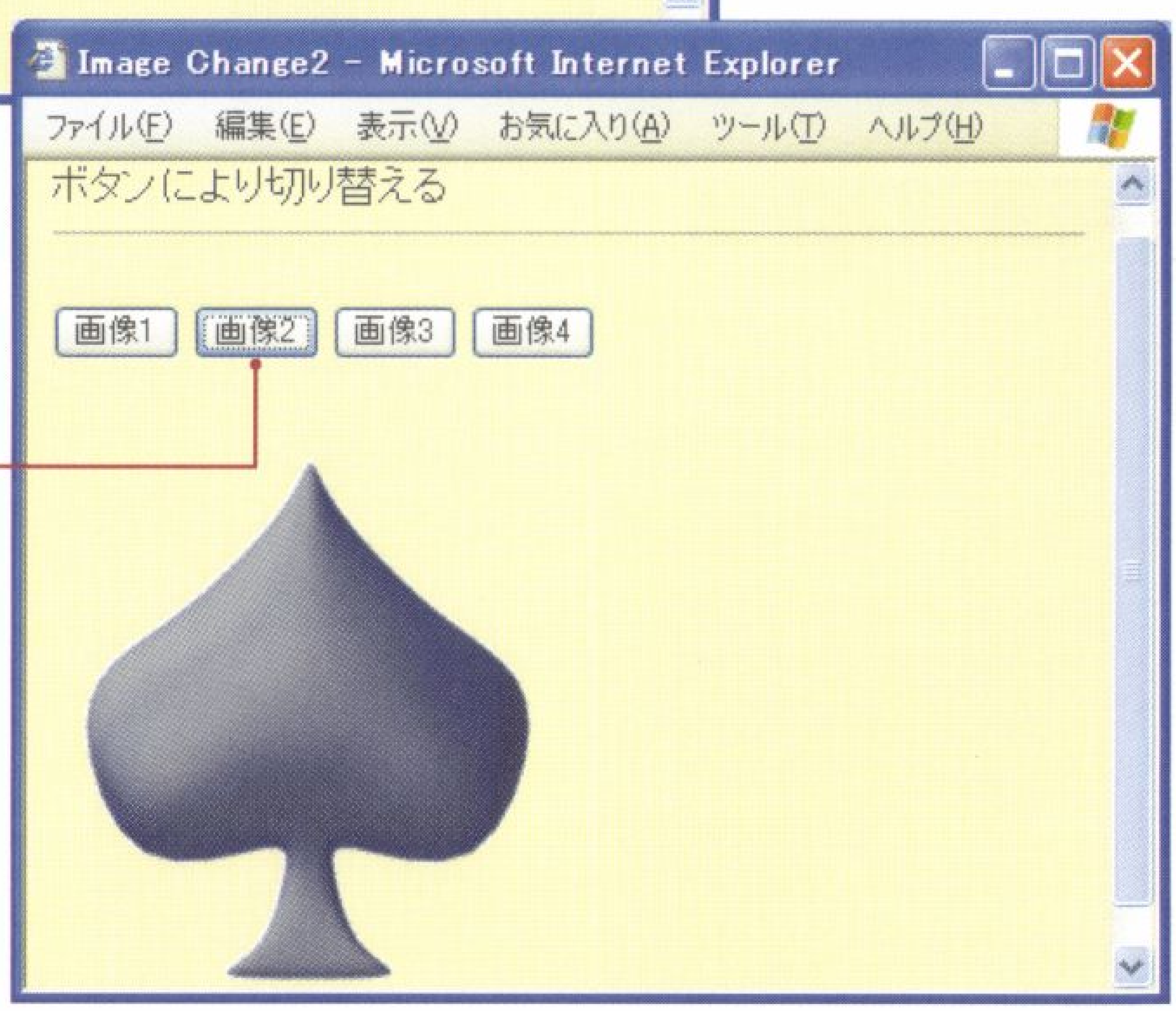
document.Imageオブジェクト名.src = "画像ファイル名"

表示されている画像を、スクリプトによって後から差し替えます。インラインでスクリプトを書いて画像を差し替えることも可能ですが、それだと複雑な処理は行えません。そのためスクリプトはヘッダ部に分離しています。これだと、複雑な処理も記述しやすく、保守性もよくなります。

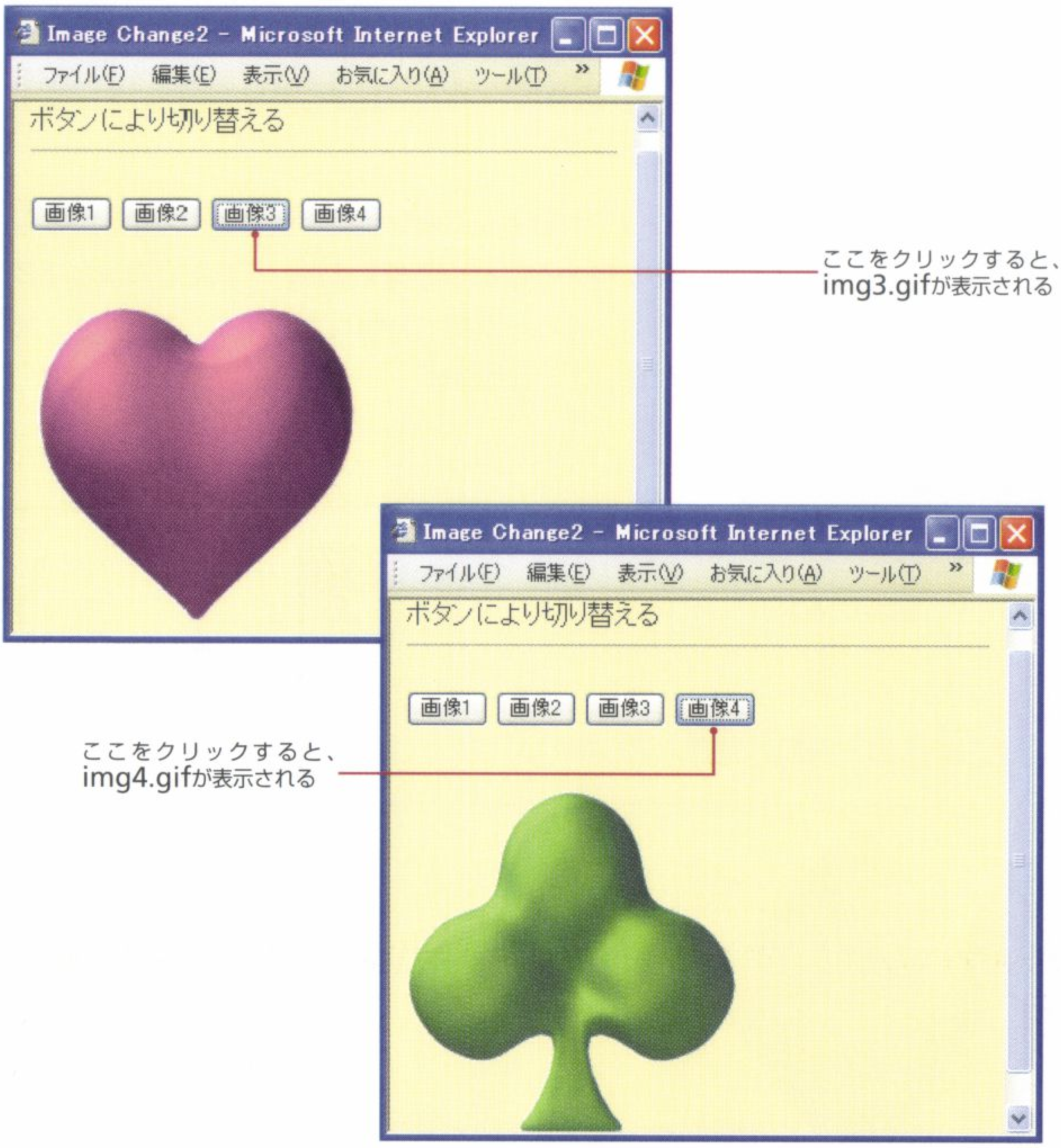
D I S P L A Y



ここをクリックすると、img1.gifが表示される



ここをクリックすると、img2.gifが表示される



SOURCE

imagechange2.html

```
<html>
<title>Image Change2</title>
<script language="javascript">
<!--
function changeImage(imageName){
   document.myImg.src = imageName;
}
//-->
</script>
<body bgcolor="#fff8dc">
ボタンにより切り替える<br />
<hr />
```

```
<form>
    <input type="button" value="画像1" onclick="
        changeImage('images/img1.gif')" />  ②
    <input type="button" value="画像2" onclick=
        "changeImage('images/img2.gif')" />
    <input type="button" value="画像3" onclick=
        "changeImage('images/img3.gif')" />
    <input type="button" value="画像4" onclick=
        "changeImage('images/img4.gif')" />
</form>
<br />
<img  name="myImg" src="images/img1.gif" />  ①
</body>
</html>
```

DETAILS

「画像1」、「画像2」、「画像3」、「画像4」と書かれたボタンをクリックすることにより画像を切り替えるスクリプトです。このスクリプトでは、汎用的に使えるようにスクリプト内(<script>〜</script>)に関数を定義し、さらに引数により切り替える画像を指定しています。

Imageオブジェクトに「myImg」という名前を付け、最初は「img1.gif」を表示するように指定しています(①の箇所)。

ボタンをクリックすると、onclickイベントハンドラによりchangeImage関数が呼び出されます。このときにクリックしたボタンによって、changeImage関数へ渡す引数が変わります。この引数により画像を指定しています。

②の部分のchangeImage関数では、変数imageNameの引数に、関数を呼び出すときに渡された画像ファイルのパスが格納されています。この変数imageNameを使って画像を変更する部分は次のようになります。

```
document.myImg.src = imageName;
```

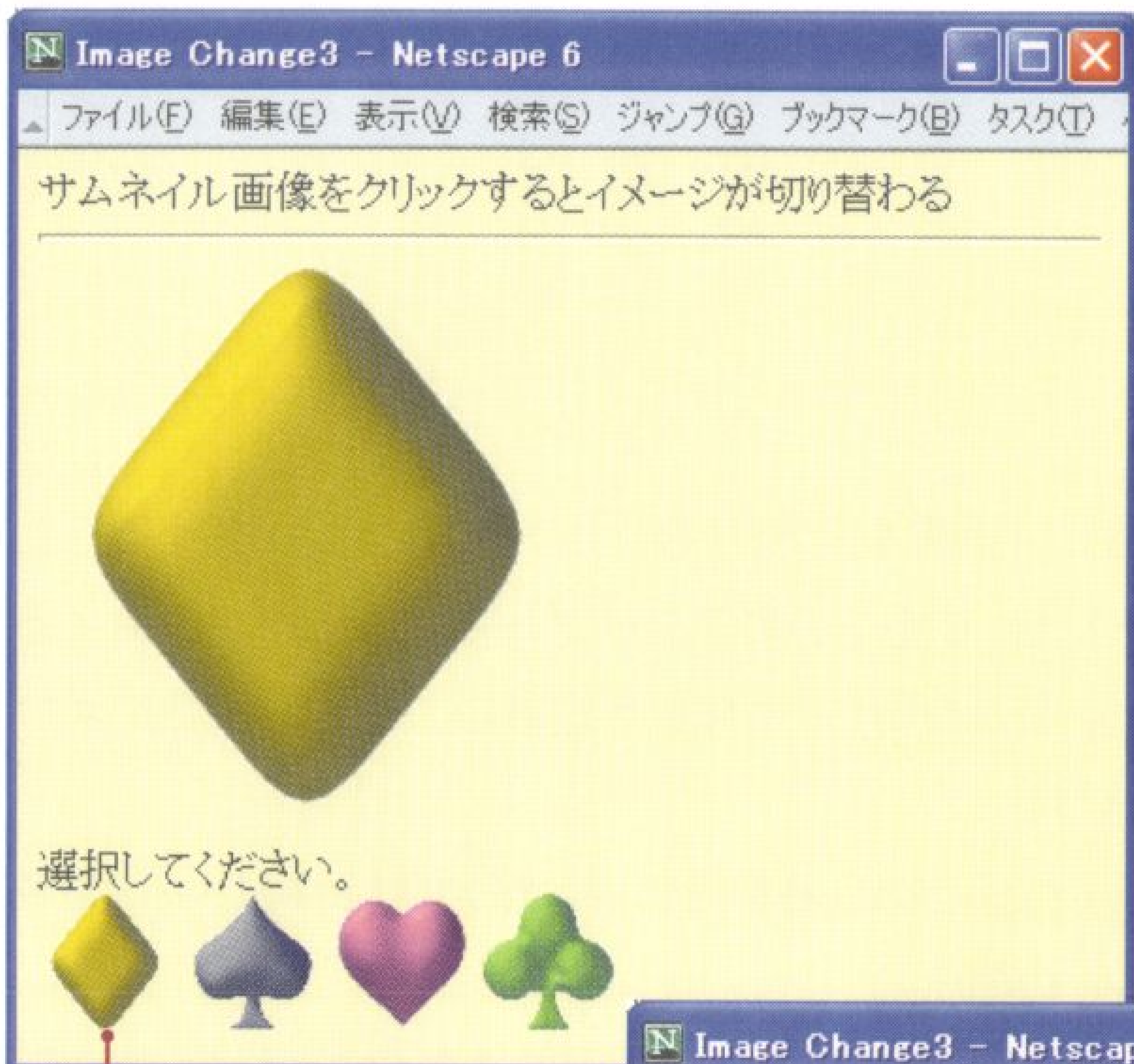
これは、documentオブジェクト(このWebページ)内にある、myImgと名付けられたImageオブジェクトのsrcプロパティに、変数imageNameに格納されているパスの画像ファイルを設定しています。srcプロパティは表示する画像ファイルを指定するものなので、これで画像が変数imageName内に格納されている画像ファイルに差し替えられます。

サムネイルから画像を選ぶ

img src="サムネイル画像" onclick="changeImage('表示する画像')"

画像を複数見せる場合に、画像サイズが大きいまま配置してしまうと、Webページが重くなってしまうばかりではなく、見た目が汚くなってしまう場合があります。このようなときには、サムネイル(縮小した画像)を用意して、ユーザーに、表示したい画像のサムネイルを選択させるスタイルを使います。

DISPLAY



このサムネイル(img1s.gif)をクリックすると、img1.gifが大きく表示される



このサムネイル(img3s.gif)をクリックすると、img3.gifが大きく表示される

SOURCE

imagechange3.html

```
<html>
<title>Image Change3</title>
<script language="javascript">
<!--
function changeImage(imageName){
   document.targetimage.src = imageName;
}
//-->
</script>
<body bgcolor="#fff8dc">
サムネイル画像をクリックするとイメージが切り替わる
<hr />
<img name="targetimage" src="images/img1.gif" /><br />
<br />
選択してください。<br />
<img src="images/img1s.gif" onclick="changeImage
➡   ('images/img1.gif')" />
<img src="images/img2s.gif" onclick="changeImage
➡   ('images/img2.gif')" />
<img src="images/img3s.gif" onclick="changeImage
➡   ('images/img3.gif')" />
<img src="images/img4s.gif" onclick="changeImage
➡   ('images/img4.gif')" />
</body>
</html>
```


DETAILS

このサンプルでは大きな画像が1つ、その下にサムネイル(小さな画像)が4枚配置されています。このサムネイルをクリックすると、大きな画像が切り替わって表示されます。

スクリプト自体は、「7-2　ボタンクリックで画像を切り替える(拡張版)」で紹介したものと、ほとんど同じです。

画像をクリックすると、<img>タグに指定されているonclickイベントハンドラにより、changeImage関数が呼び出されます。クリックしたボタンによって、changeImage関数へ渡す引数が変わります。この引数で、表示する画像を指定

しています。

changeImage関数では、変数imageNameを使って画像を変更しています。ドキュメントの階層構造にしたがって大きな画像を指定し、そのsrcプロパティに変数imageNameに格納されているファイル名を設定します。

N O T E S

このスクリプトは、NN4では動作しません。それは、NN4では<img>タグ内のonclickイベントハンドラを扱うことができないからです。NN4でも動作させたい場合には、サムネイルの部分を次にように記述しておきます。

```
<a href="javascript: changeImage('images/img1.gif')">
⇨ <img src="images/img1s.gif" border="0" /></a>
<a href="javascript: changeImage('images/img2.gif')">
⇨ <img src="images/img2s.gif" border="0"/></a>
<a href="javascript: changeImage('images/img3.gif')">
⇨ <img src="images/img3s.gif" border="0"/></a>
<a href="javascript: changeImage('images/img4.gif')">
⇨ <img src="images/img4s.gif" border="0"/></a>
```

このように記述すると、アンカーのクリックによりJavaScriptの命令が呼び出され、changeImage関数を実行します。

これでNN4でも、サムネイルをクリックして、大きな画像が表示されるようになります。

7-4 WIN 4↑ N 4↑ MAC 4↑ N 4↑

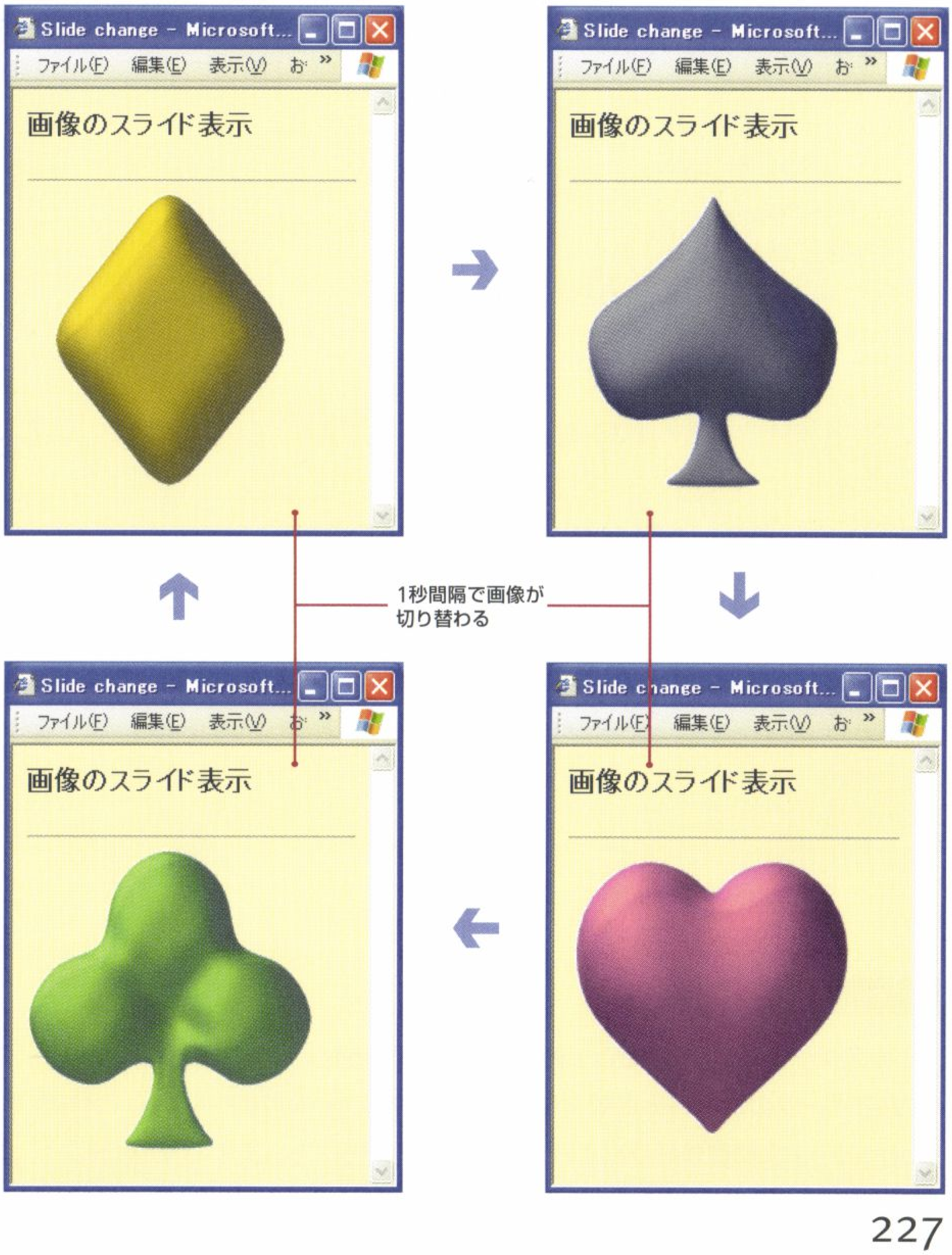
画像を一定時間で切り替える

setTimeout("実行する命令", 時間)

document.Imageオブジェクト名.src="画像ファイル名"

バナー広告のように、複数の画像が一定の間隔で切り替わっていくスクリプトです。限られたスペースで複数の画像を見せたい場合などに効果的でしょう。このサンプルでは、自動的に4枚の画像を、1秒間隔で表示するために使っています。

DISPLAY



SOURCE imagechange4.html

```
<html>
<title>Slide Change</title>
<script language="javascript">
<!--
var imgNum=1;                                          ②

function changeImage(){
    if(imgNum==1){
        document.myImg.src="images/img2.gif";          ③
        imgNum=2;
    }else if(imgNum==2){
        document.myImg.src="images/img3.gif";          ④
        imgNum=3;
    }else if(imgNum==3){
        document.myImg.src="images/img4.gif";
        imgNum=4;
    }else if(imgNum==4){
        document.myImg.src="images/img1.gif";
        imgNum=1;
    }
    setTimeout("changeImage()",1000);
}
//-->
</script>
<body onload="changeImage()" bgcolor="#fff8dc">        ①
<h3>画像のスライド表示</h3>
<hr />
<img src="images/img1.gif" name="myImg" />
</body>
</html>
```

DETAILS

このスクリプトでは1秒ごとに画像が次々に切り替わっていきます。

Webページの読み込みと同時に<body>タグのonloadイベントハンドラでchangeImage関数が呼び出され、スクリプトが実行されます（①の箇所）。

また、Webページが読み込まれた時点で「imgNum」という変数に「1」が代入されます（②の箇所）。そのため、changeImage関数が最初に呼び出されたときには、変数imgNumの値は「1」に設定されています。

if文で変数imgNumが「1」ならば、myImgというイメージオブジェクトのsrcプロパティを「img2.gif」にして、変数imgNumに2を代入します。ここで、画像は③の部分で設定されている「img1.gif」から「img2.gif」に切り替わります。画像を切り替えるif-else文を抜けると次の命令が実行されます。

```
setTimeout("changeImage()",1000);
```

setTimeoutメソッドは、一定時間後に指定された関数や命令を実行するためのメソッドです。ここでは、1000ミリ秒後、つまり1秒後に、再びchangeImage関数を呼び出しています。この時点で、imgNumに格納されている値は「2」になります。そのため、次に呼び出されたchangeImage関数では、④の部分が実行されることになります。そこで再び画像と変数imgNumの値が変更され、changeImage関数が呼び出されます。これを繰り返していくことによって、4枚の画像が次々と切り替わっていきます。

TIPS

このサンプルをご覧になって、「これならアニメーションGIFを使えばいいじゃないか」と思われる人もいるでしょう。事実、Webページの広告で用いられているのはアニメーションGIFがほとんどです。しかしアニメーションGIFのサイズ（容量）は決して小さくないため、すべての人の環境のことを考えると、大きなものはなるべく控えるべきでしょう。

そこで、アニメーションにこだわらない場合は、JavaScriptによるイメージの切り替えを使うとよいと思います。1回スクリプトを記述してしまえば、アニメーションGIFに比べてさまざまなバリエーションを作ることができます（切り替えるイメージの名前を変えるだけですから）。またプリロードなどを使うことにより、ユーザーにとってストレスのないページ表示を実現しやすくなります。さらに、この方法を使えば、複数のアニメーションGIFを切り替えて表示することもできます。

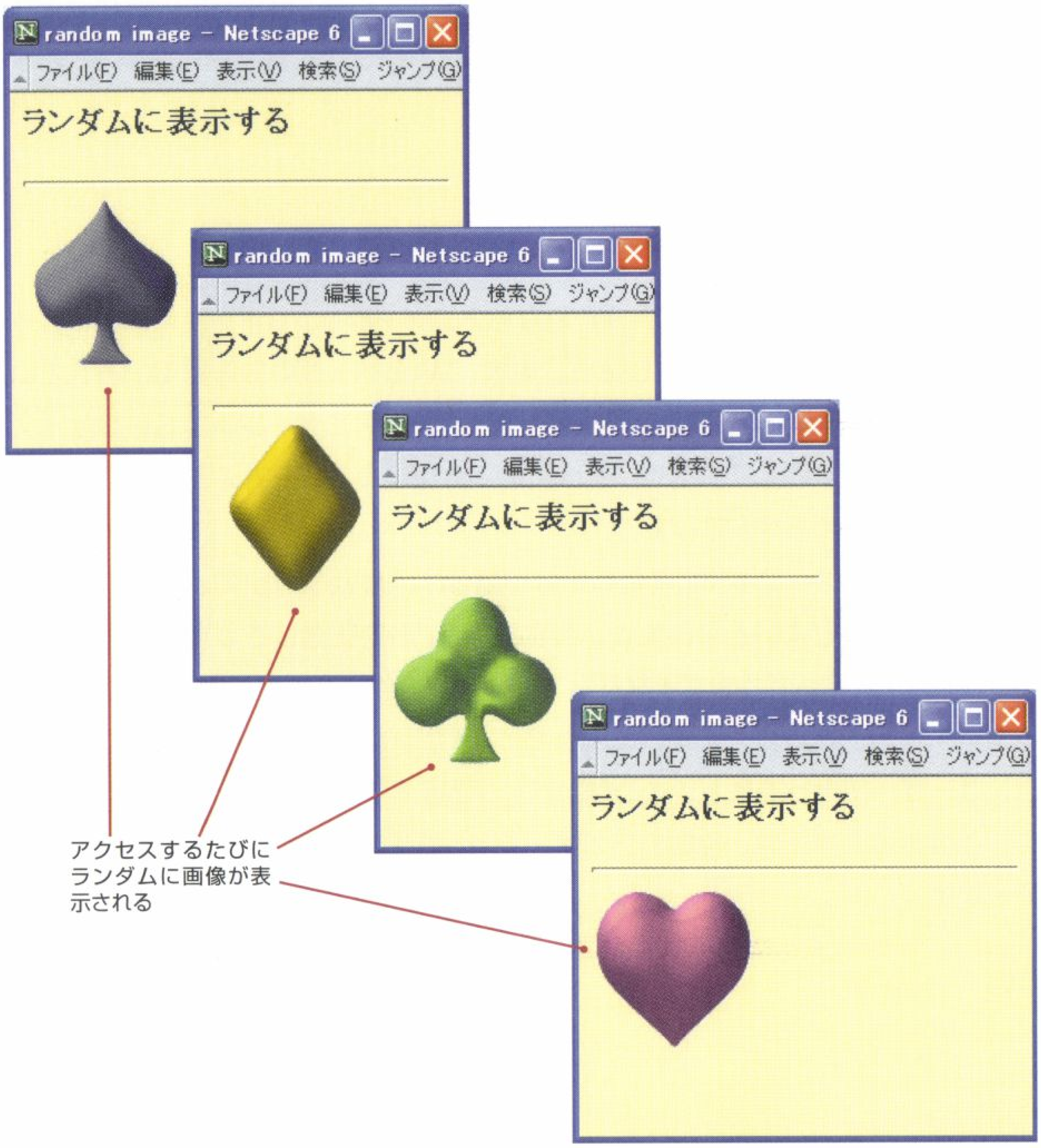
7-5　WIN e 4↑ N 4↑　MAC e 4↑ N 4↑

画像をランダムに表示する

Math.random()*項目数
Math.ceil(整数に切り上げたい数値);
eval("連結する文字列" + 数値);

いつも同じWebページではつまらないものです。とはいえ、毎日更新するのも大変です。そんなときに、アクセントがあって、ページのイメージが変わるのはよいかもしれません。ページの画像を変更するだけでも、かなり印象が変わってくるはずです。ここでは、Webページの読み込み時に表示される画像をランダムに変えてみましょう。

DISPLAY



IMAGE

SOURCE

randamimage.html

```
<html>
<title>random image</title>
<script language="javascript">
<!--
var iValue, rNumber;

var img1 = new Image();
var img2 = new Image();
var img3 = new Image();
var img4 = new Image();
img1.src = "images/img1.gif";
img2.src = "images/img2.gif";
img3.src = "images/img3.gif";   ①
img4.src = "images/img4.gif";

function displayImage(){
   iValue = Math.random() * 4;   ③
   rNumber = Math.ceil(iValue);
   if(rNumber==0) rNumber++;   ④
   document.image1.src=eval("img" + rNumber + ".src");   ⑤
}
//-->
</script>
<body onload="displayImage()" bgcolor="#fff8dc">
<h3>ランダムに表示する</h3>
<hr />
<img name="image1" width="100" height="100" />   ②
</body>
</html>
```

DETAILS

このサンプルでは4つの画像(img1.gif、img2.gif、img3.gif、img4.gif)を用意し、Webページを読み込むたびにランダムに表示しています。

スクリプトの最初で、表示する画像を扱うImageオブジェクト(img1、img2、img3、img4)を用意します。この作成したImageオブジェクトに対して画像フ

ァイルのURIを指定します(①の箇所)。これにより画像が格納され、Imageオブジェクトが利用できるようになります。Imageオブジェクトはsrc属性にURIを指定すると、表示されていない画像でもWebブラウザにダウンロードします。

次に、画像を表示する位置を指定します。ここでは、<img>要素を配置し、それにname属性でimage1と名付けています(②の箇所)。src属性は、後からスクリプトにより指定できるので記述していません。画像をランダムに選択し表示する機能は、displayImage関数で実装しています。displayImage関数は、<body>タグのonloadイベントハンドラでWebページのロード時に呼び出されています。

さて、画像をランダムに表示するとき、どうやって決定しているかというと、組み込みで乱数を発生する、Mathオブジェクトのrandomメソッドを使います。

```
Math.random()
```

randomメソッドは0から1の範囲(0、1も含みます)で乱数を返します。これを使ってランダムに選択するという機能を作ります。表示したい画像が、たとえば4つだったら、乱数で発生した数値に4をかけます(③の箇所)。すると、0から4の範囲でさまざまな数値が現れます。しかし、このままでは選択肢としては使えないので、Mathオブジェクトのceilメソッドで数値を整数に切り上げます。ceilメソッドは、数値自身よりも大きい一番近い整数に切り上げます。

発生した数値	切り上げた数値
0	0
0より大きく1以内	1
1より大きく2以内	2
2より大きく3以内	3
3より大きく4以内	4

その結果、表の5つの整数になります。0はほとんど現れることがありませんが、一応エラー処理もかねて変数rNumberが「0」のときには「1」にするという処理を記述して4つの整数にします(④の箇所)。

最後にこの選択された数値を用いて、Imageオブジェクトを指定します。それにはevalメソッドを用いて、「img」という文字列とrNumberに格納された数値を連結します。すると、img1、img2、img3、img4のいずれかのImageオブジェクトを表します。これを事前に用意した、<img>要素に対して代入することによりイメージを表示します。これらを行っているのが⑤の箇所です。

7-6

WIN e4↑ N6↑　MAC e4↑ N6↑

選択した画像の説明をテキストボックスに表示する

document.formオブジェクト名.elementオブジェクト名.value="表示するテキスト"

画像の上をマウスカーソルが通過したときに、テキストボックスに説明が表示されるスクリプトです。いくつか画像を用意しておいて、ユーザーが知りたいと思った情報だけを表示させることが可能です。

DISPLAY



画像の上にマウスカーソルを置くと、上のテキストボックスにその説明が表示される

SOURCE imagewithcomment.html

```
<html>
<title>Image with Comment</title>
<script language="javascript">
<!--
function showMsg(imageNo){
    if(imageNo==1){
        document.myForm.myText.value = "ダイア";
    }
    else if(imageNo==2){
        document.myForm.myText.value = "スペード";
    }
    else if(imageNo==3){
        document.myForm.myText.value = "ハート";
    }
    else if(imageNo==4){
        document.myForm.myText.value = "クローバー";
    }
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>画像のコメントを表示する</h3>
<hr />
画像の上を通過すると・・・<br />
<form name="myForm">
<input type="text" name="myText" size="20" value="画像の説明" />
</form>
<br />
<img src="images/img1.gif" onmouseover="showMsg(1)" />
<img src="images/img2.gif" onmouseover="showMsg(2)" /><br />
<img src="images/img3.gif" onmouseover="showMsg(3)" />
<img src="images/img4.gif" onmouseover="showMsg(4)" />
</body>
</html>
```

①

DETAILS

このサンプルでは、4つの画像が並んで配置され、その上にテキストボックスが配置されています。マウスカーソルが各画像の上を通過すると、テキストボックスにその画像に割り当てた説明が表示されます。

画像の上をマウスカーソルが通過すると、<img>タグのonmouseoverイベントハンドラによってshowMsg関数が呼び出されます。そして、showMsg関数では引数により、どの画像から呼び出されたのかを特定しています。画像と番号の関係は、「DISPLAY」画面で、左上のダイアが1、右上のスペードが2、左下のハートが3、右下のクローバーが4になります。

showMsg関数では、引数で受け取った数値が格納されている変数imageNoにより条件分岐を行います（①の箇所）。この条件分岐によりどのカーソルがどの画像上にあるかを特定します。特定ができたら次のようにして、テキストボックス内にテキストを表示させます。

```
document.myForm.myText.value = "スペード";
```

テキストボックスの中に表示される文字列は、value属性で設定できます。

このサンプルでは、「myForm」という名前のformオブジェクト内にある、「myText」という名前のelementオブジェクトのvalueプロパティにさまざまなテキストを代入しています。

他の画像の上をマウスカーソルが通過したときにも、それぞれ呼び出される関数によって同様の処理が行われます。この結果、画像の上をマウスカーソルが通過することによって、フォームに表示されるメッセージが切り替わります。

TIPS

このスクリプトはNN4では動作しません。それはNN4では、<img>タグ内でonmouseoverイベントハンドラを処理できないためです。そこで、イメージの部分のコードの記述を、次のように変更すると動作するようになります。

```
<a href="void(0)" onmouseover=""showMsg(1)">
<img src="images/img1.gif" />
</a>
```

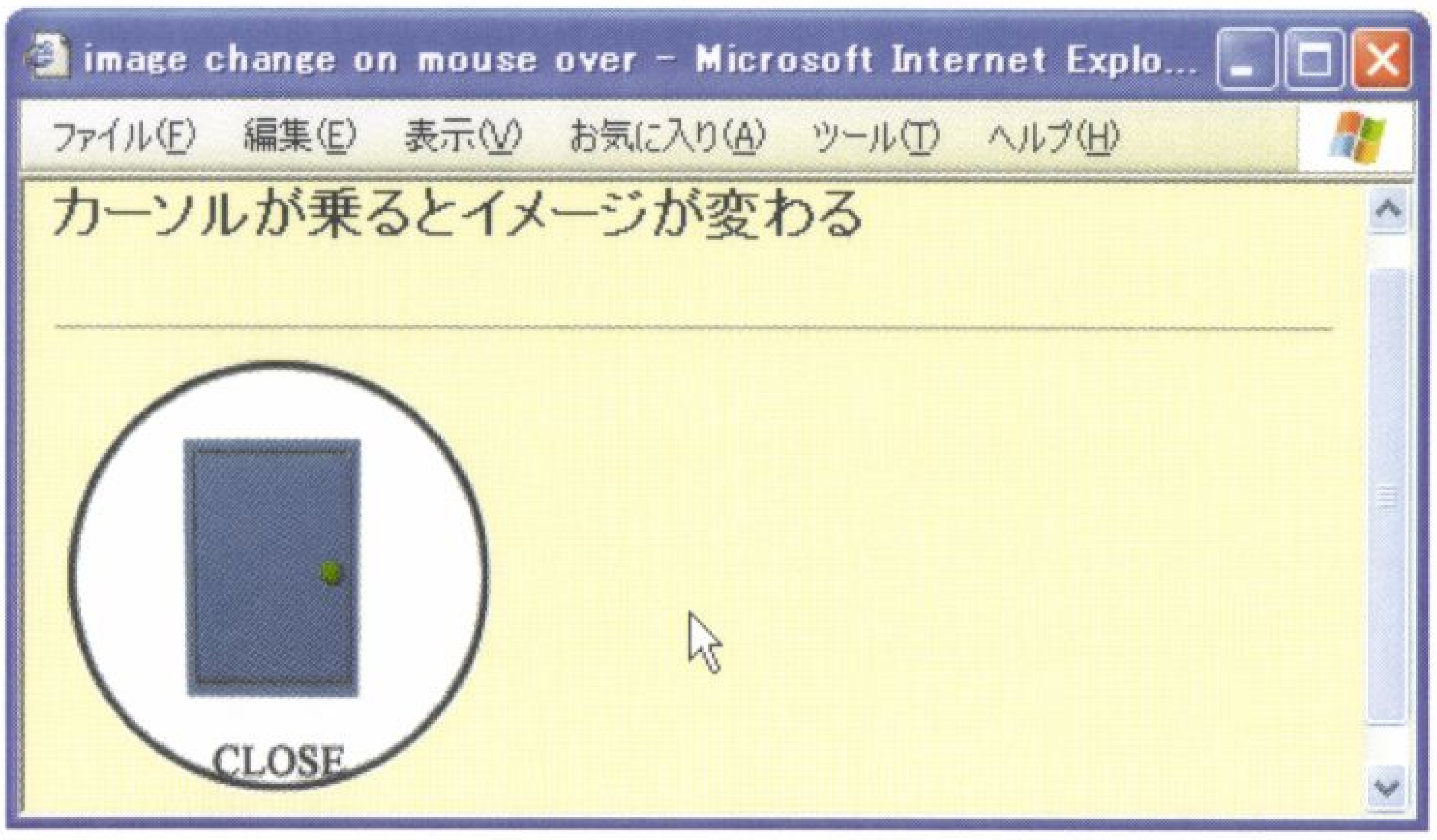
マウスオーバーで画像が切り替わる

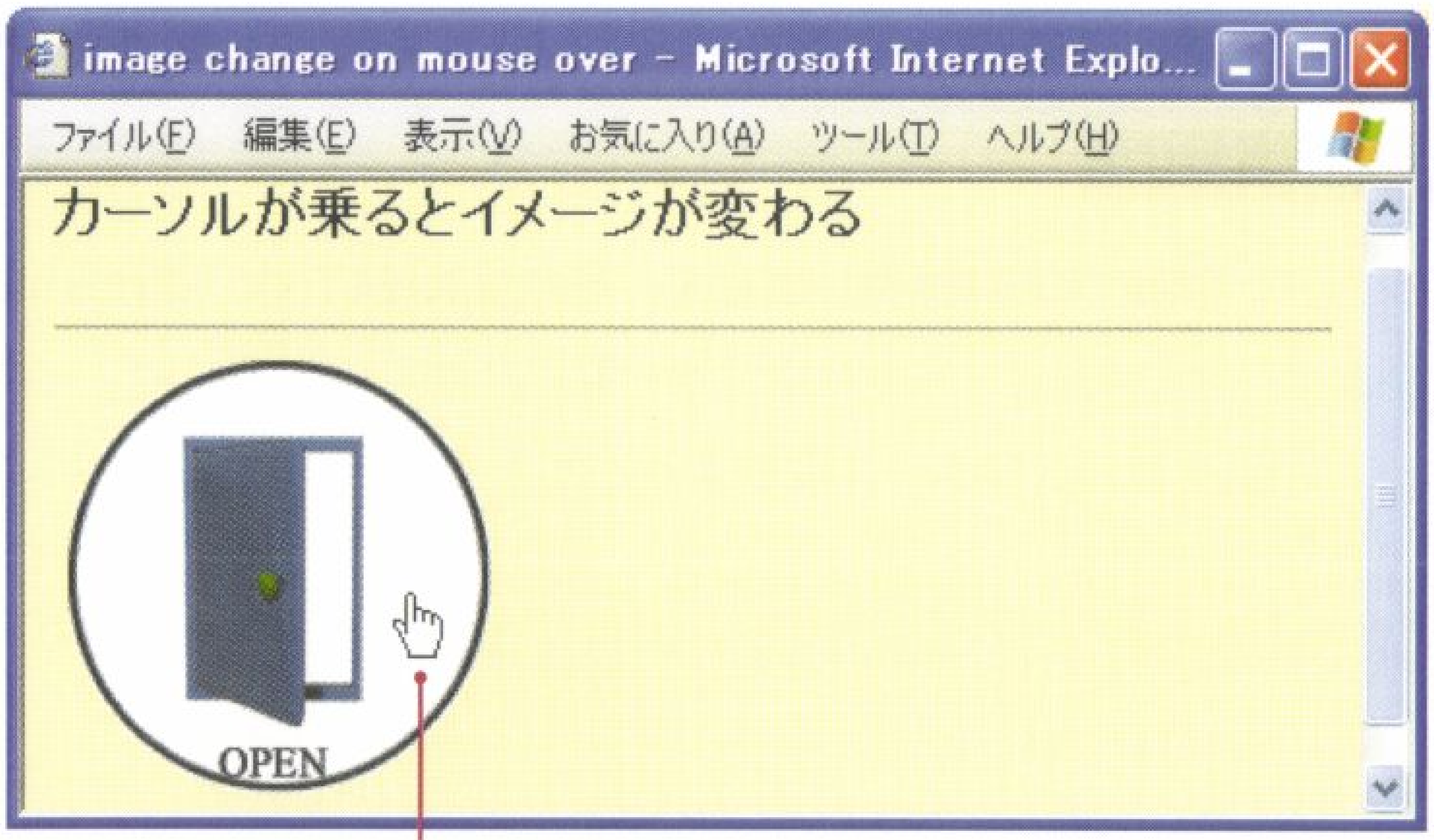
```
var Imageオブジェクト名 = new Image()
```

```
Imageオブジェクト名.src = "画像ファイル名"
```

マウスカーソルが重なると画像が切り替わる動作は、Webページを閲覧しているユーザーに対して非常に効果的な表現です。自分がどこにマウスカーソルを合わせているのかもわかりますし、ユーザーがクリックしたくなるという点では非常に有効なスクリプトです。

D I S P L A Y





マウスカーソルを画像の
上に置くと、ドアが開く

SOURCE rollonimage1.html

```
<html>
<title>image change on mouse over</title>
<script language="javascript">
<!--
var img1_on = new Image();
var img1_off = new Image();
img1_on.src="images/imgOpen.gif";              ①
img1_off.src="images/imgClose.gif";

function rollon(targetimage){
   if(targetimage=="image1"){
      document.images[targetimage].src=img1_on.src;   ②
   }
}

function rolloff(targetimage){
   if(targetimage=="image1"){
      document.images[targetimage].src=img1_off.src;  ③
   }
}
// -->
</script>
<body bgcolor="#fff8dc">
<h3>カーソルが乗るとイメージが変わる</h3>
<hr />
<a href="test.html">
   <img name="image1" src="images/imgClose.gif" border="0"
➡      onmouseout="rolloff('image1')" onmouseover=
➡      "rollon('image1')" />
</a>
</body>
</html>
```


DETAILS

このサンプルではImageオブジェクトを使っています。これは、マウスカーソルが画像に重なったときに変化する画像がすぐに表示できるように、画像をプリ

ロードしておくためです。プリロードを組み込むには、次の書式で記述します。

```
var <Imageオブジェクトを扱うための変数> = new Image();
```

最初にImageオブジェクトの実体を生成し、画像を扱うための変数(名前は任意)を用意します。ここでは「img1_on」と「img1_off」の2つを用意しています。次に、Imageオブジェクトに対して画像を次の書式で設定しています。

```
Imageオブジェクト名.src = "画像ファイル名"
```

Imageオブジェクトのsrcプロパティに、実際の画像ファイル名を指定すると自動的に画像が取り込まれます(①の箇所)。プリロードは、Webページ読み込み時に行わないと意味がありませんので、Webページ読み込み時に実行されない関数内で、プリロードするファイルを指定しないよう注意してください。

次に画像を切り替える部分です。画像にカーソルが重なるとonmouseoverイベントハンドラによりrollon関数が呼び出され、カーソルが画像から外れたときにonmouseoutイベントハンドラによりrolloff関数が呼び出されます。この2つの動作により2枚の画像を切り替えています(②と③の箇所)。これらの関数内ではImageオブジェクトのsrcプロパティに、事前に用意しておいたImageオブジェクトを代入することにより、画像を切り替えています。引数を渡しているのは、複数の画像を処理するのに対応するためです。ここで引き渡している引数は、どの<img>要素を変更するかを指定するためです。

T I P S

このサンプルのスクリプトは、NN4では動作しません。NN4では、<img>タグ内でonmouseoverイベントハンドラ、onmouseoutイベントハンドラを処理できないのです。NN4では、イメージの部分のコードを、次のように変更します。

```
<a href="test.html"
⇒  onmouseover="rollon('image1')"
⇒  onmouseout="rolloff('image1')">
      <img  name="image1"  src="images/imgClose.gif"
⇒        border="0" />
</a>
```

WIN e 4↑ N 6↑ MAC e 5↑ N 6↑

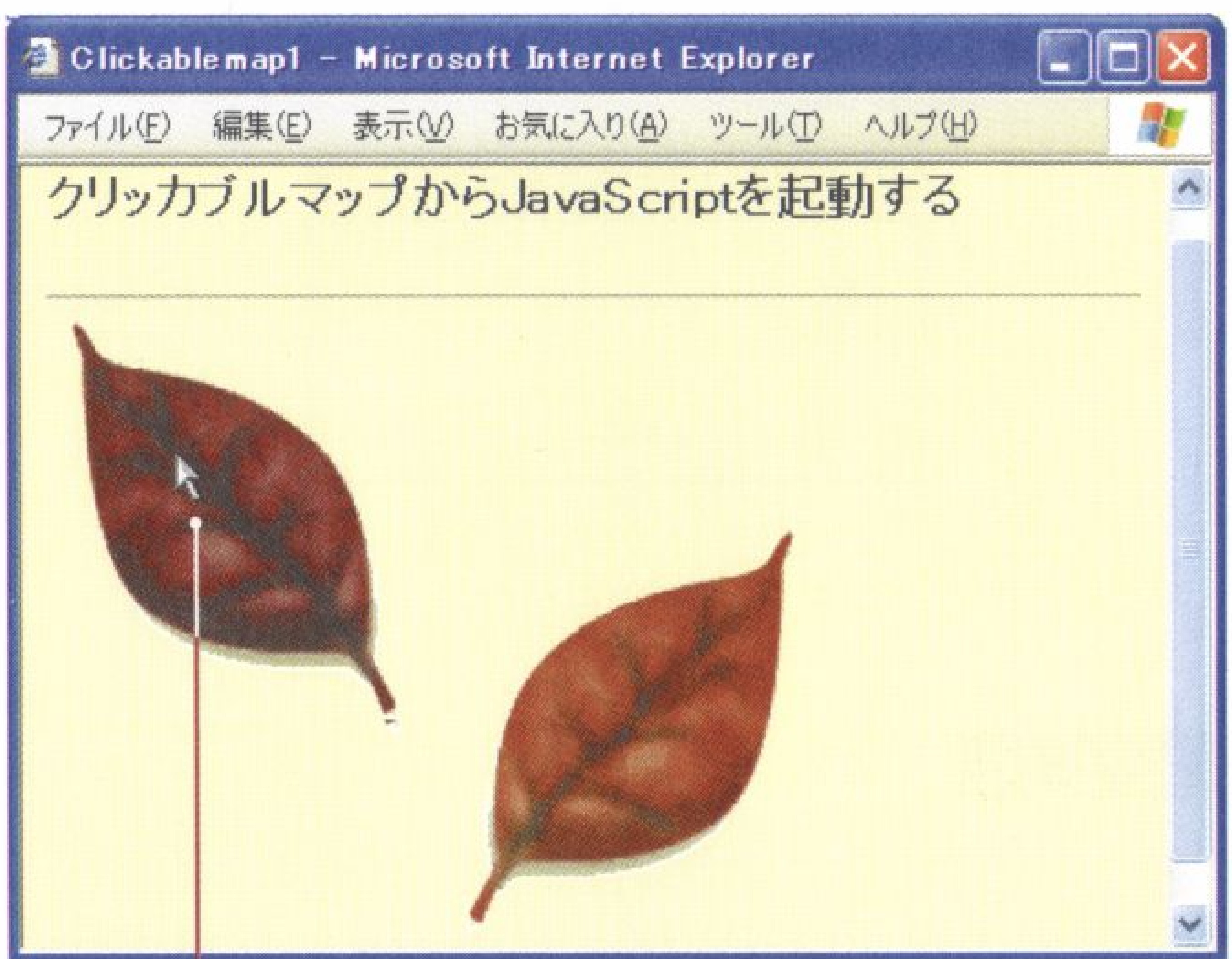
クリッカブルマップ

```
<area … onmouseover = "呼び出す命令">
<area … onClick = "呼び出す命令">
<area … href="javascript:呼び出す命令">
```

クリッカブルマップというのは、現在のWebページの制作においてなくてはならない手法です。1枚の画像に対して複数のクリックを設定できるのは、デザイン性の高いメニューなどにおいては必須です。通常、クリッカブルマップにはリンクしか設定できませんが、JavaScriptを使うとさらに複雑な処理を付加することができます。

D I S P L A Y

▼マウスカーソルを置いてアラートを表示する

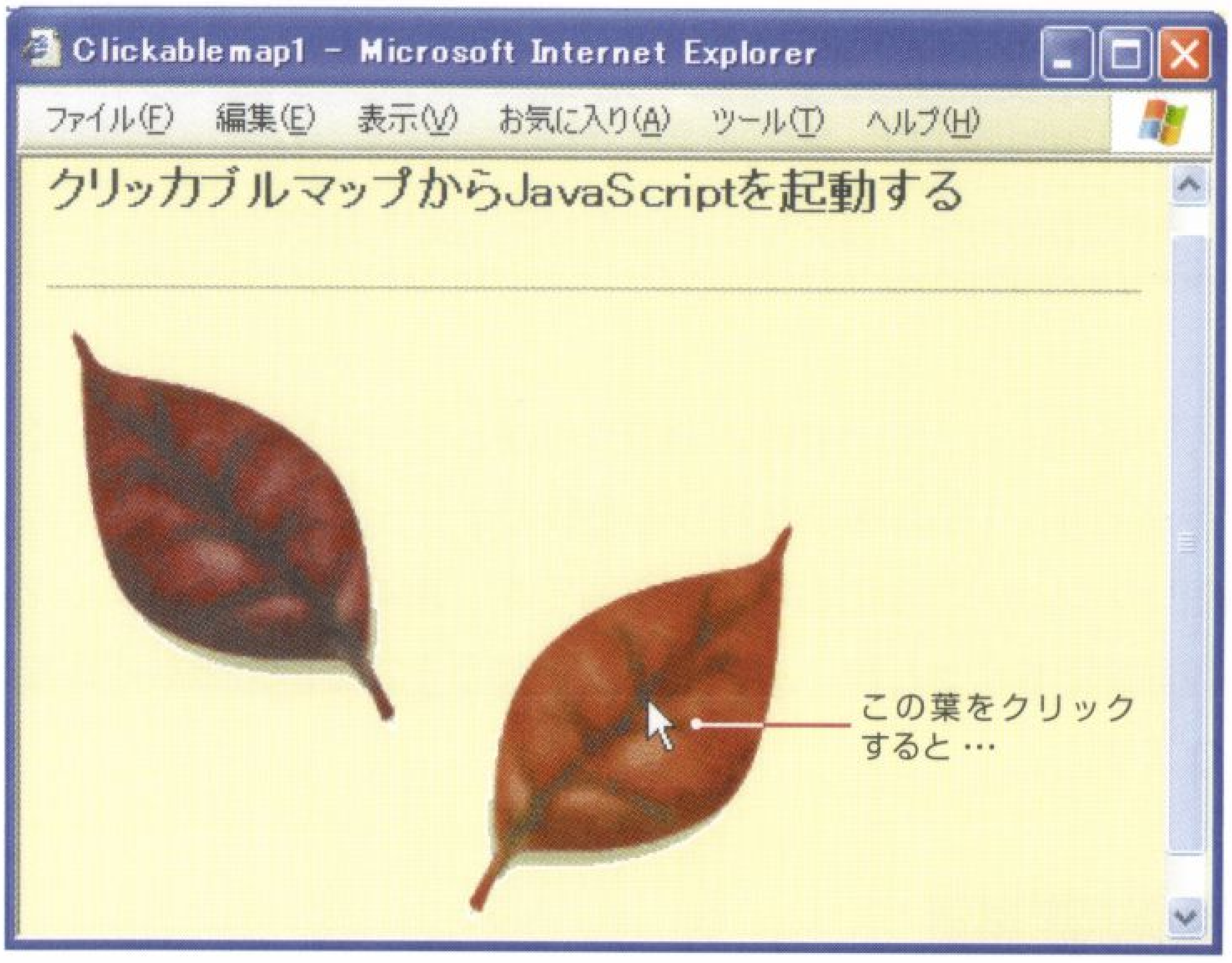


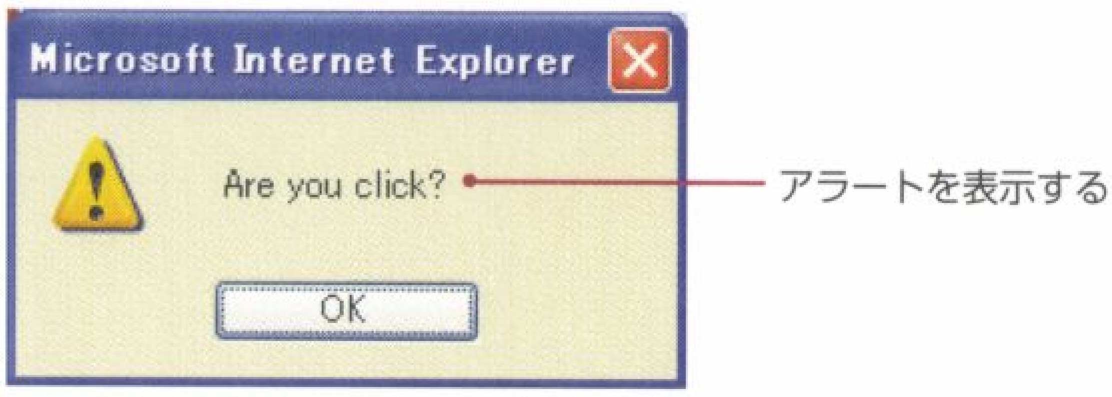
この葉の上にマウスカーソルを置くと…



アラートを表示する

▼マウスクリックでアラートを表示する





SOURCE

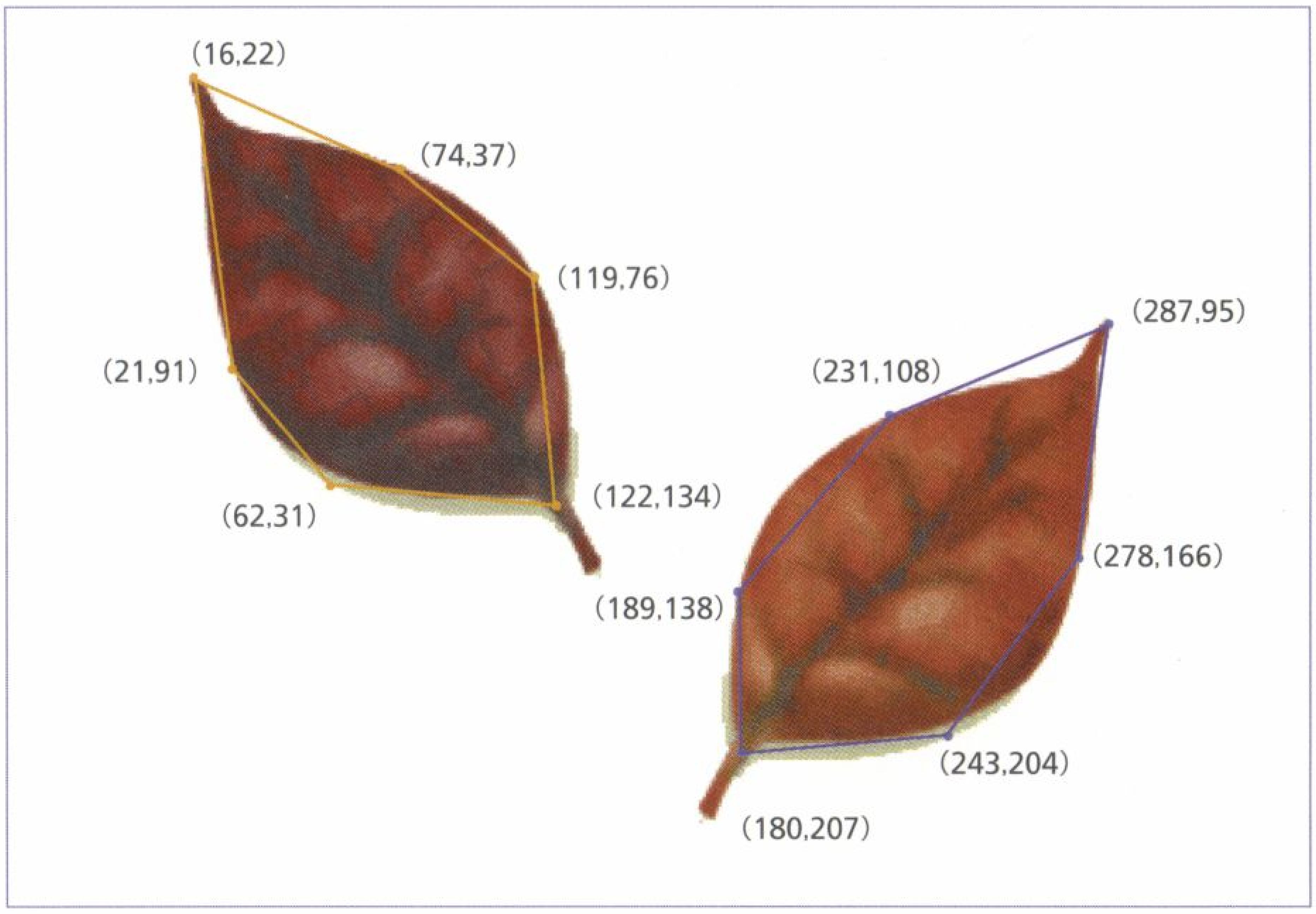
clickablemap.html

```
<html>
<title>clickablemap1</title>
<script language="javascript">
<!--
function displayAlert(char_message){
   alert(char_message);
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>クリッカブルマップからJavaScriptを起動する</h3>
<hr />
```

```
<img src="images/clickablemap.gif" usemap="#map1" border="0" />
<map name="map1">
    <area shape="polygon" coords="16,22,21,91,62,131,
        122,134,119,76,74,37" onmouseover="displayAlert(        ①
        'Are you mouseover?')" />
    <area shape="polygon" coords="180,207,243,204,278,166,
        287,95,231,108,189,138" onclick="displayAlert(          ②
        'Are you click?')" />
</map>
</body>
</html>
```


DETAILS

クリッカブルマップの中では、通常、他のWebページのリンクを張ることしかできません。しかし、イベントやJavaScriptを使うと、リンクを張る以外の機能を実現することができます。サンプルでは、次のようなクリッカブルマップを用意しています。



クリッカブルマップからJavaScriptを呼び出すには、<area>要素にイベントハンドラを記述します。左上の葉の部分にはonmouseoverイベントハンドラを

(①の箇所)、右下の葉にはonclickイベントハンドラを設定しています(②の箇所)。このイベントハンドラからdisplayAlert関数を呼び出しています。ここでは、アラートを出力しているだけなので関数にする意味はあまりありませんが、実際の場面ではさまざまな処理を行うことが考えられるため、イベントハンドラから関数の呼び出しを行っています。

T I P S

このサンプルでは、<area>要素に対してイベントハンドラを使っているためにNN4では動作しません。また、Macintosh版IE4.5でも動作しません。これは<area>要素においてイベントハンドラを用いることができないためです。

しかし、次のようにイベントハンドラを使わないスクリプトなら、クリッカブルマップのリンクからJavaScriptの命令を呼び出すことができます。

clickablemap_nn4.html

```
<html>
<title>Clickablemap2</title>
<script language="javascript">
<!--
function displayAlert(char_message){
   alert(char_message);
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>クリッカブルマップからJavaScriptを起動する NN4</h3>
<hr />
<img src="images/clickablemap.gif" usemap="#map1"
⇒  border="0" />
<map name="map1">
   <area shape="polygon" coords="16,22,21,91,62,131,122,
⇒     134,119,76,74,37" href="javascript: displayAlert
⇒     ('Are you click from NN4?')" />
   <area shape="polygon" coords="180,207,243,204,278,166,
⇒     287,95,231,108,189,138" href="javascript:
⇒     displayAlert('Are you click from NN4?')" />
```

```
</map>
</body>
</html>
```

このサンプルで注目するのは<area>要素のhref属性の部分です。本来はリンク先を書くべきところに、次の書式で記述しています。

```
href="javascript: 呼び出す命令"
```

このようにすると、JavaScriptの命令を呼び出すことができます。命令の代わりに関数を呼び出すこともできます。サンプルでは、実際にさまざまな機能を追加することができるように、関数を定義して呼び出しています。

ただし、このようにしてしまうと、残念ながらクリックによるイベントしか取れなくなってしまいます。

■クリッカブルマップの作成

クリッカブルマップをWebページに実装するには、次のような記述が必要になります。

```
<img src="画像名" usemap="#マップ名">
<map name="マップ名">
<area shape="リンクの形" href="リンク先" />
...
</map>
```

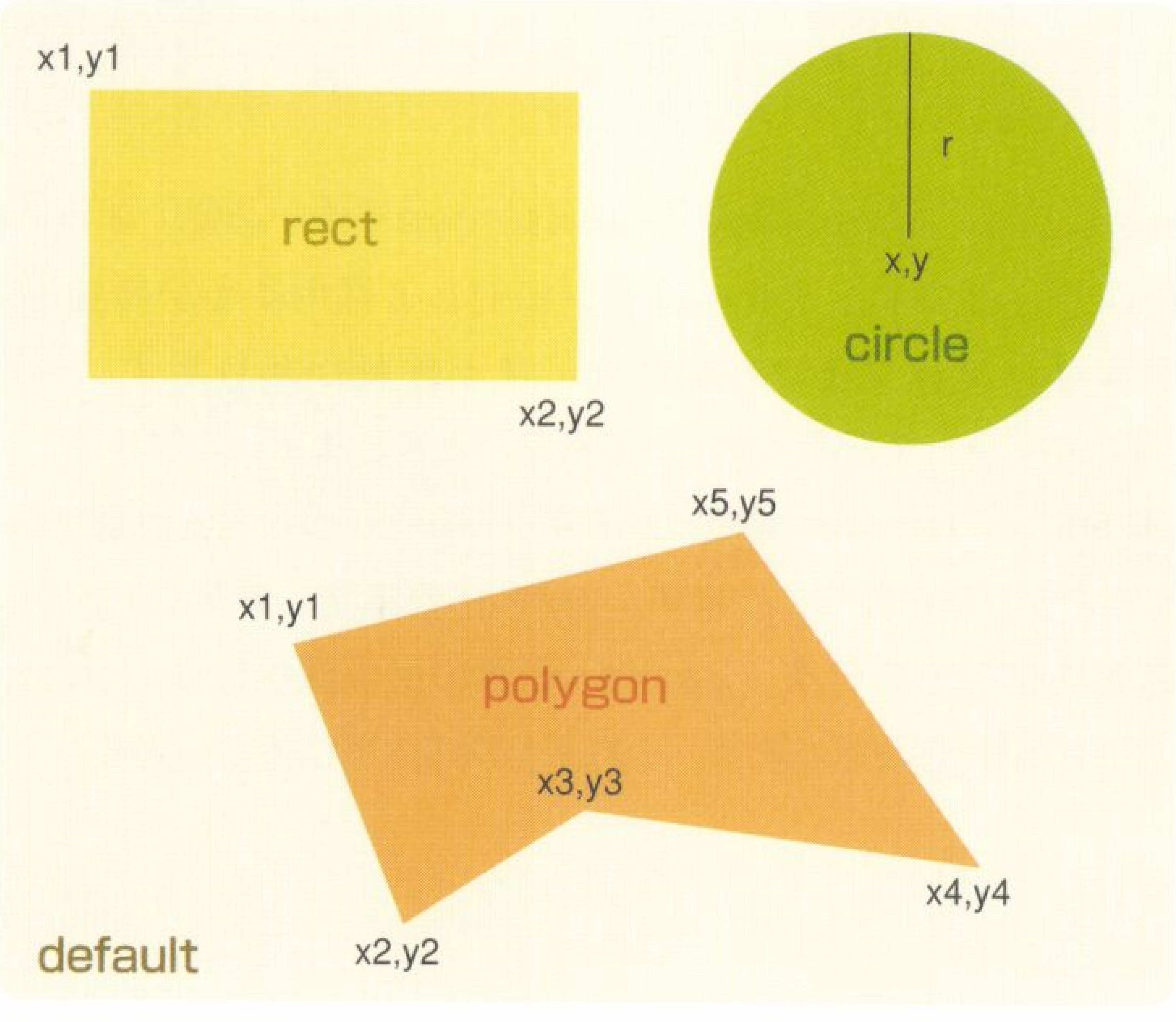
ポイントとしては、<img>要素の中にusemap属性が必要になることです。この属性により、どのマップをイメージに対して適用するかを指定します。<map>要素は、マップをひとまとめにする役割があります。実際には、<map>要素の中に入っている<area>要素をひとまとめにしています。

<map>要素には、usemap属性でどのマップかということを区別するために、name属性でマップに名前を付けます。<area>要素は、実際にイメージ上にリンクする範囲を作るものです。<area>要素の主な属性には、shape属性、coords属性、href属性があります。それぞれ次のように指定します。

属性	説明	設定値
shapes	リンク範囲の形状を指定する	rect、長方形
		circle、円
		polygon、多角形
		default、リンクが指定されていない部分
coords	リンク範囲の形状にしたがって座標を指定する	座標、半径など
href	リンク先を指定する	絶対座標、相対座標
alt	代替テキストを指定する	代替文字列
nohref	リンクがないことを宣言する	

なお、coords属性の指定はshapes属性に何を指定したかで変わります。それぞれの対応を書くと次のようになります。

shapes属性	coords属性
rect	x1, y1、x2, y2
circle	x, y、r
polygon	x1, y1、x2, y2、………、xN, yN
default	なし



JavaScript
Handbook
3rd edition

8-1

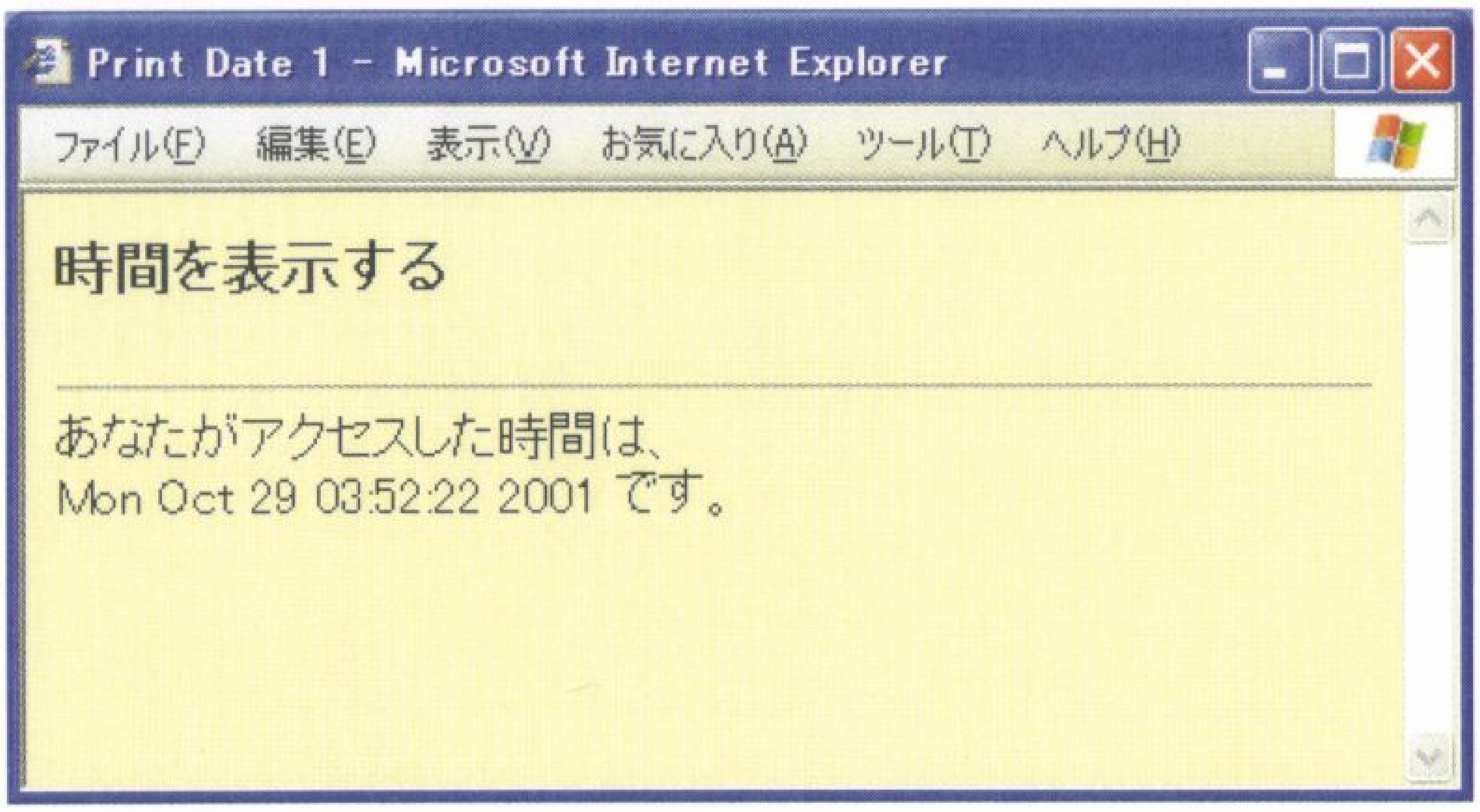
WIN IE 4↑ N 4↑　MAC IE 4↑ N 4↑

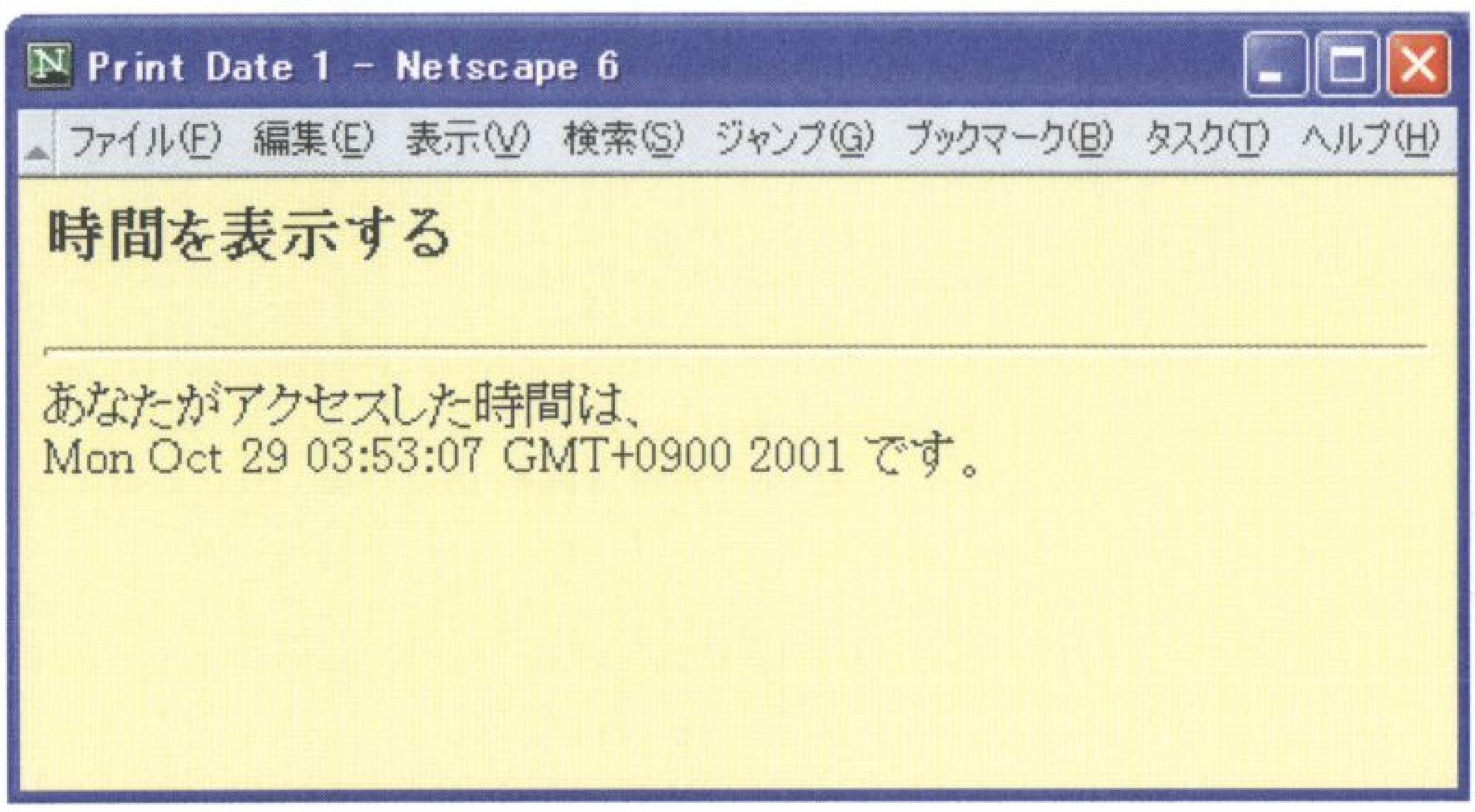
アクセスした日付を表示する

Date()

アクセスした日付・時刻を表示するスクリプトです。「ただいま＊月＊日＊時＊分です」といった表示をする場合によく使われます。このスクリプトでは表示形式は指定していないため、コンピュータが返す形式で表示しています。また、IEとNNでも表示形式が違います。

D I S P L A Y





SOURCE printdate1.html

```
<html>
<title>Print Date1</title>
<body bgcolor="#fff8dc">
<h3>時間を表示する</h3>
<hr />
あなたがアクセスした時間は、<br/>
<script language="javascript">
<!--
   document.write(Date());
//-->
</script>
です。
</body>
</html>
```

DETAILS

重要となるのは、「Date()」と記述しているDateオブジェクトです。これはシステムが持っている日付・時刻を返します。ここで返ってくる日付・時刻はローカルマシン、要するに自分のマシンに設定してあるものです。

```
document.write(Date());
```

上記の記述では返ってきた日付・時刻を画面に表示します。documentオブジェクトのwriteメソッドは次のような書式です。

```
document.write(表示するテキスト)
```

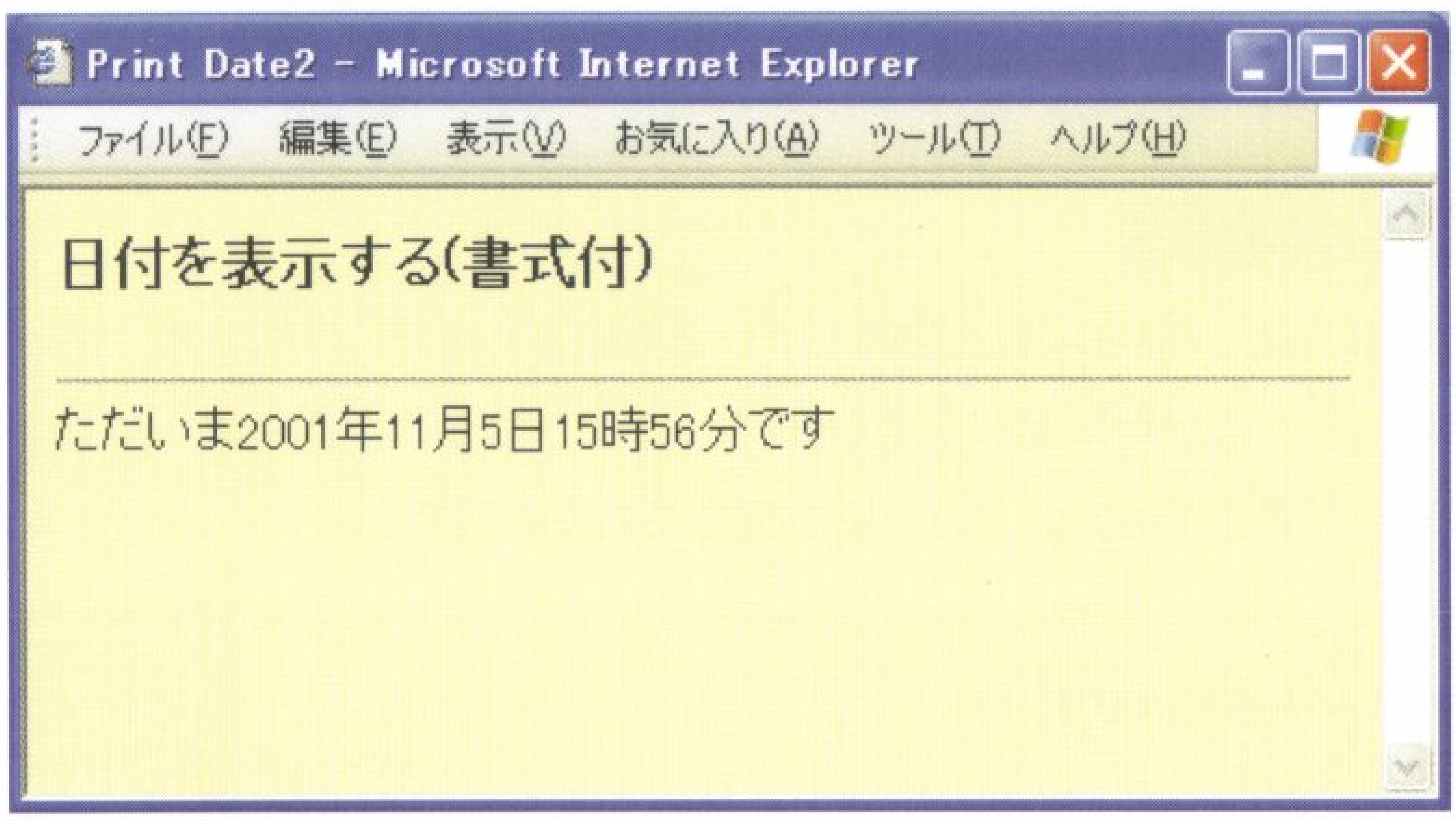
注意点として、IEとNNでは返ってくる日付・時刻の形式が異なります。これではWebページのデザイン上困ることがありますので、「8-2　日付時間の書式を指定して表示する」で紹介する、書式を指定するスクリプトも併せてご覧ください。

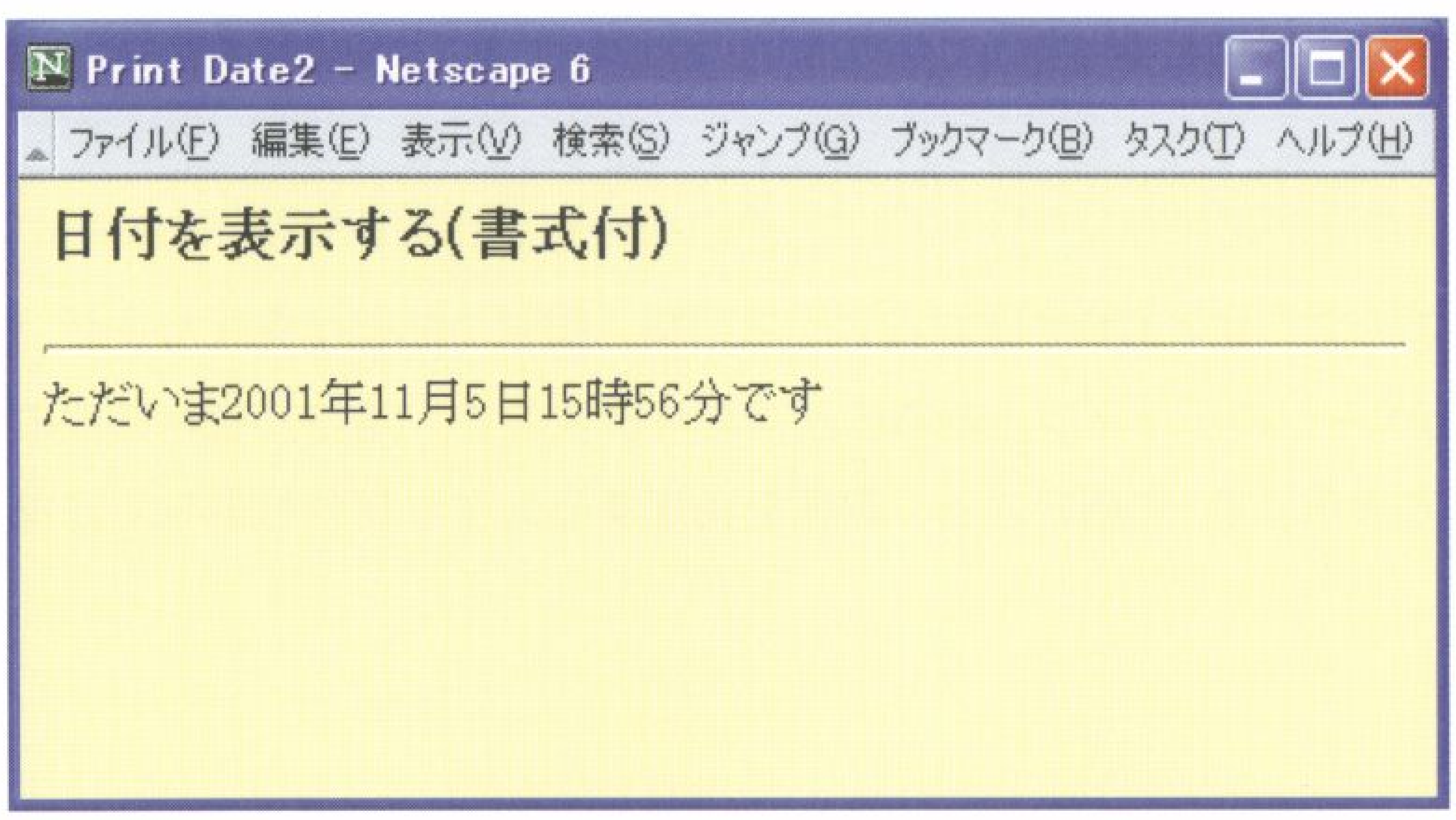
日付時間の書式を指定して表示する

```
Dateオブジェクト名=new Date()
Dateオブジェクト名.getYear()
Dateオブジェクト名.getMonth()
Dateオブジェクト名.getDate()
Dateオブジェクト名.getHours()
Dateオブジェクト名.getMinutes()
```

アクセスしてきた、日付・時刻を表示するときに、システムが返してきたままの書式で表示するのは味気がないものです。また、IEとNNでは形式が異なる問題もあります。そこで、日付・時刻に書式の指定をして表示するようにします。

DISPLAY





SOURCE printdate2.html

```
<html>
<title>Print Date2</title>
<script language="javascript">
<!--
mySysDate=new Date(); ──①
myYear=mySysDate.getFullYear();
myMonth=mySysDate.getMonth();
myDate=mySysDate.getDate();
myHour=mySysDate.getHours()
myMinutes=mySysDate.getMinutes();
//-->
</script>
<body bgcolor="#fff8dc">
<h3>日付を表示する(書式付)</h3>
<hr />
<script language="javascript">
<!--
  document.write("ただいま",myYear,"年",myMonth+1,"月",myDate,
⇒     "日",myHour,"時",myMinutes,"分です");
//-->
</script>
</body>
</html>
```

DETAILS

このスクリプトでは、まず①の部分でDateオブジェクトを生成しています。

```
mySysDate = new Date();
```

生成したDateオブジェクト「mySysDate」から年、月、日、時、分などを、それぞれメソッドを使って抜き出します。主なメソッドには、次のようなものがあります。

取り出す単位	メソッド
年	getFullYear()、getYear()
月	getMonth()
日	getDate()
時	getHours()
分	getMinutes()
秒	getSeconds()
曜日	getDay()

この中で注意が必要なのが、getMonthメソッドでは月が「0～11」の数字で抜き出されることです。そのため、表示するときは、月の値が格納されている変数に「1」を足さないと正しい月が表示されません。

また、西暦の扱いに関してですが、getYearメソッドでは、1999年までは西暦の下2桁を、2000年以降は4桁で西暦を返し、getFullYearメソッドはすべて4桁の西暦を返します。ただし、getFullYearはIE4、NN4以降でしか対応していないため、使うときは注意してください。

getYearメソッドは、下2桁と4桁を混在しているため、2000年をまたがる西暦を扱い、比較などを行うと、正しく処理できない場合があります。NN4の一部のバージョンでは、2000年を100年と表示してしまいますので、これにも注意が必要です。

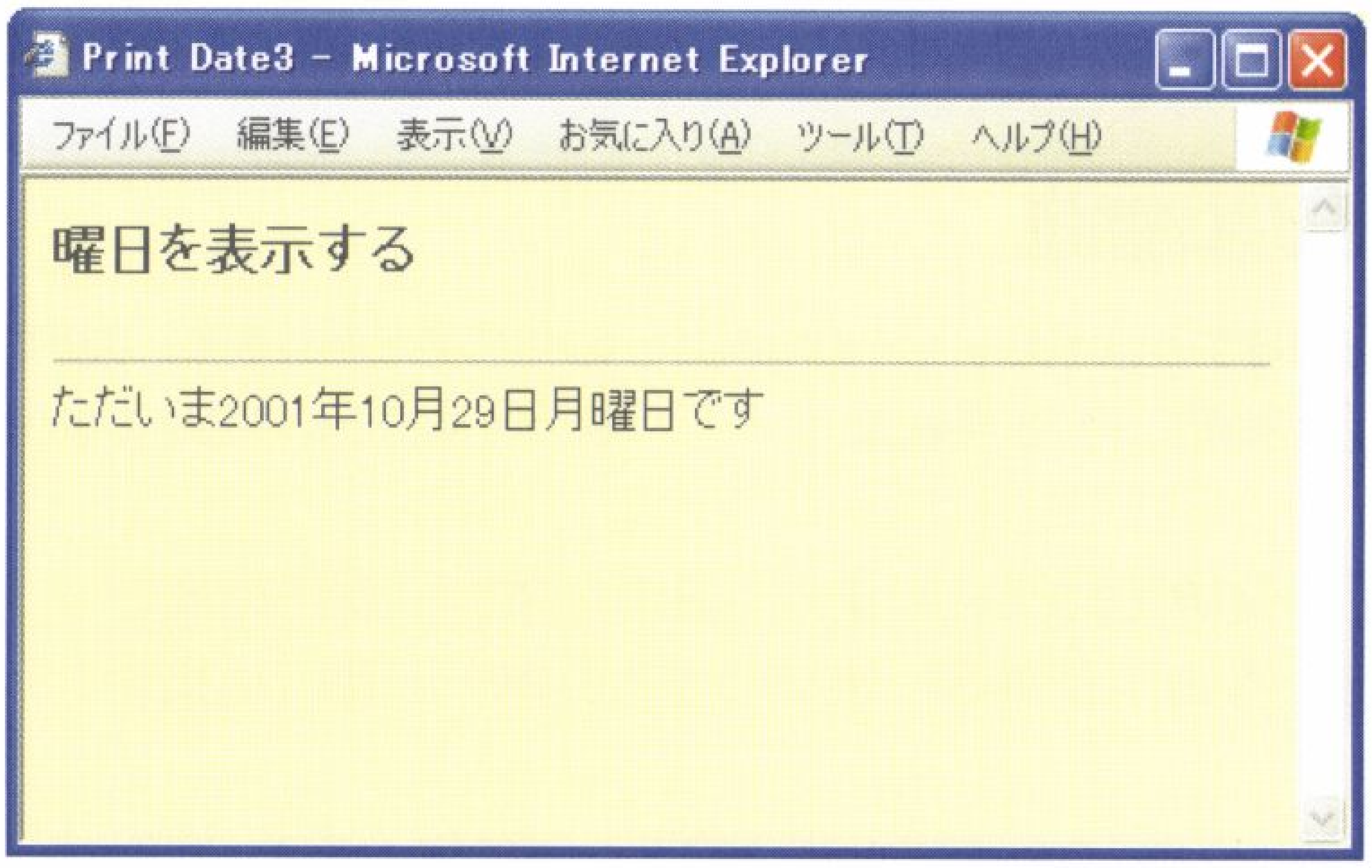
曜日を表示する

Arrayオブジェクト名 = new Array(配列の要素)

Dateオブジェクト名.getDay()

曜日を出力するスクリプトです。日本語で曜日を出力するには、ちょっとした変換が必要になります。日付を出すのに比べて、そのための処理を少し加えなくてはなりません。

DISPLAY



SOURCE

printdate3.html

```
<html>
<title>Print Date3</title>
<script language="javascript">
<!--
mySysDate=new Date();
myYear=mySysDate.getFullYear();
myMonth=mySysDate.getMonth();
myDate=mySysDate.getDate();
day=new Array("日","月","火","水","木","金","土"); ──①
myDay=day[mySysDate.getDay()];
//-->
</script>
```

```
<body bgcolor="#fff8dc">
<h3>曜日を表示する(書式付)</h3>
<hr />
<script language="JavaScript">
<!--
   document.write("ただいま",myYear,"年",myMonth+1,"月",myDate,
➡      "日",myDay,"曜日です");
//-->
</script>
</body>
</html>
```

DETAILS

曜日を取得するgetDayメソッドは「0～6」の数字を返します。次の表のように、「0」が日曜日、「1」が月曜日、………、「6」が土曜日ということになります。

数値	0	1	2	3	4	5	6
曜日	日	月	火	水	木	金	土

このため、返された数字を曜日に変換する必要があります。方法としては、if文を使って、次のように記述する方法があります。

```
myDay = mySysDate.getDay();
if(myDay == 0){myDay = "日";}
else if(myDay == 1){myDay = "月";}
else if(myDay == 2){myDay = "火";}
……
```

このように記述してもよいのですが、もう少しスマートに書くと、このサンプルのように、配列(Array)を使うことになります。Arrayオブジェクトを使って配列(サンプルではday配列)を用意します。Arrayオブジェクトは、①の部分のようにnewを使って生成します。

```
day=new Array("日","月","火","水","木","金","土");
```

Arrayオブジェクトを生成するときに、曜日の一覧を配列に代入しています。配列の添え字は「0」からはじまり、さらにgetDayメソッドで戻ってくる値の「0」は日曜日に対応するので、配列の要素を「日、月、火、水、木、金、土」としています。

ここでday配列から読み出すには、以下のような書式で行います。

```
day[添え字]
```

day配列には、次のような値が入っています。

配列	day[0]	day[1]	day[2]	day[3]	day[4]	day[5]	day[6]
値	日	月	火	水	木	金	土

ここで、getDayメソッドは数値を返してくるのでは、読み出し次のようになります。

```
myDay = day[mySysDate.getDay()];
```

この部分が数値に変わる

後は、このようにして得られたmyYear、myMonth、myDate、myDayをdocument.writeメソッドにより表示させれば完成です。

年月日の情報の取得・表示については「8-2　日付時間の書式を指定して表示する」のサンプルと変わりませんので、そちらを参照してください。

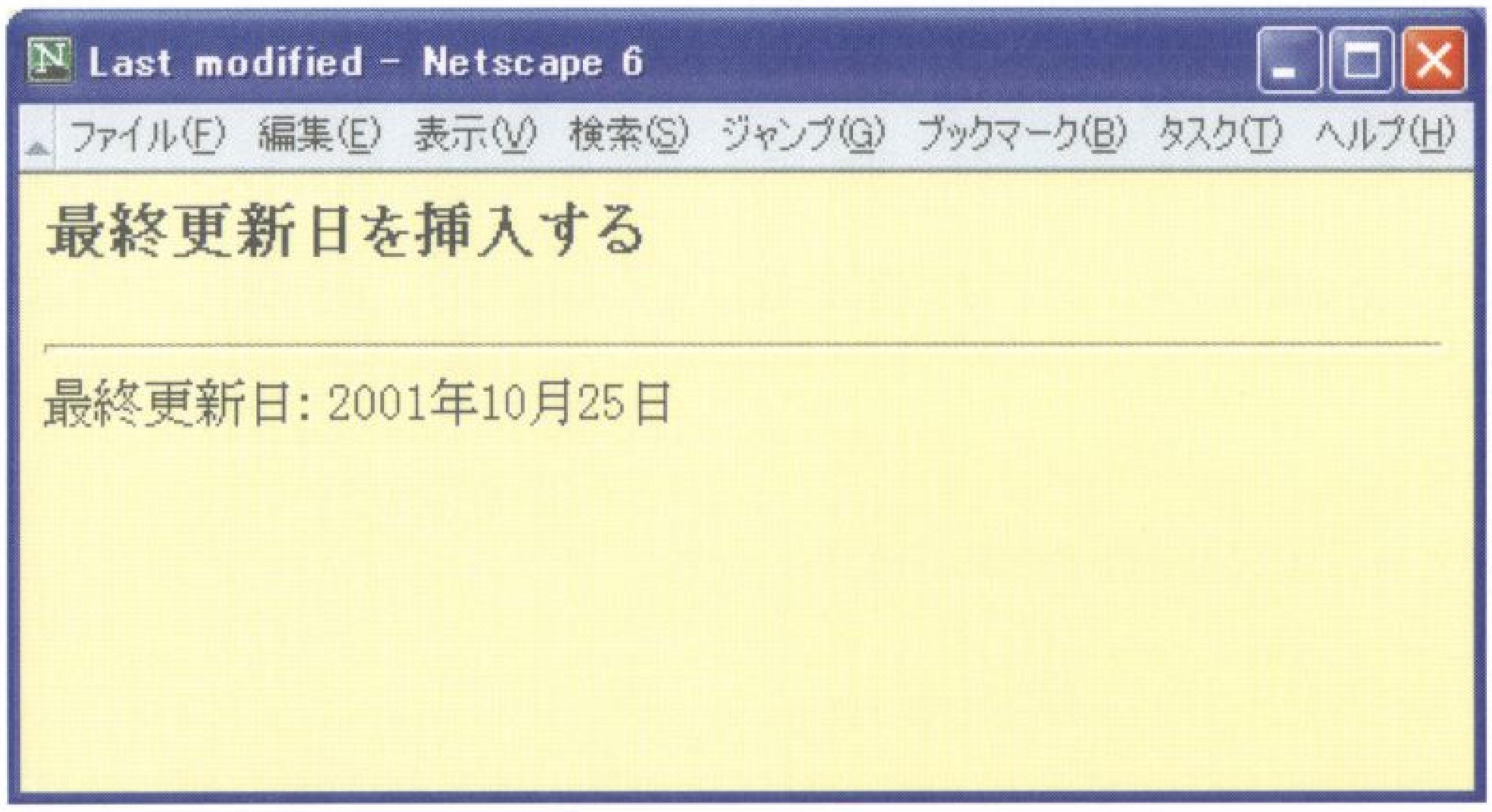
8-4 WIN 5↑ N 4↑ MAC 5↑ N 6↑

最終更新日を挿入する

document.lastModified

Webページでよく見かけるものとして、最終更新日というものがあります。コンテンツが増えてくると、ユーザーは更新されたページがあるかどうかがわかりにくくなってきます。そんなときに、トップページに最終更新日を入れておくと、更新されたかどうかがすぐにわかります。

DISPLAY



SOURCE

lastmodified.html

```
<html>
<title>Last Modified</title>
<body bgcolor="#fff8dc">
<h3>最終更新日を挿入する</h3>
<hr />
<script language="javascript">
<!--
   var lmDateStr = new Date(document.lastModified); ——①
   var lmYear = lmDateStr.getFullYear();
   var lmMonth = lmDateStr.getMonth() + 1;
   var lmDate = lmDateStr.getDate();
   document.write("最終更新日: " + lmYear + "年" + lmMonth +
⇒      "月" + lmDate + "日<br />");
//-->
```

```
</script>
</body>
</html>
```

DETAILS

このスクリプトでは、ファイルの最終更新時間を最終更新日として扱っています。そのため、まずファイルの最終更新時間を取得しています。ファイルの最終更新時間は、documentオブジェクトのlastModifiedプロパティに格納されています。

このプロパティを直接表示しても確かに最終更新時間がわかるのですが、それでは表示書式をコントロールすることができません。そのために、①の部分のようにDateオブジェクトの引数として与えます。すると、最終更新日を格納したDateオブジェクトが生成されます。

このDateオブジェクトから年、月、日を取り出すメソッドを用いて個別に値を取り出し、画面上に表示しています。

TIPS

document.lastModifiedプロパティはサーバー上の時間を返します。そのため、さまざまなトラブルが発生する可能性があります。Webサーバーによってはdocument.lastModifiedプロパティを返さないものがあります。

また、Webブラウザでも正しく動作しないものが多数あります。現在、動作が確認できるものでは、Windows上ではIE5以降、NN4以降、Macintosh上ではIE5以降、NN6以降でしか正しく動作しないようです。

そのため、インターネット上で公開するファイルにこのスクリプトを用いるのはあまりお勧めしません。対象となるWebブラウザが限定できるイントラネットなどで使うのがよいと思います。

8-5

WIN MAC

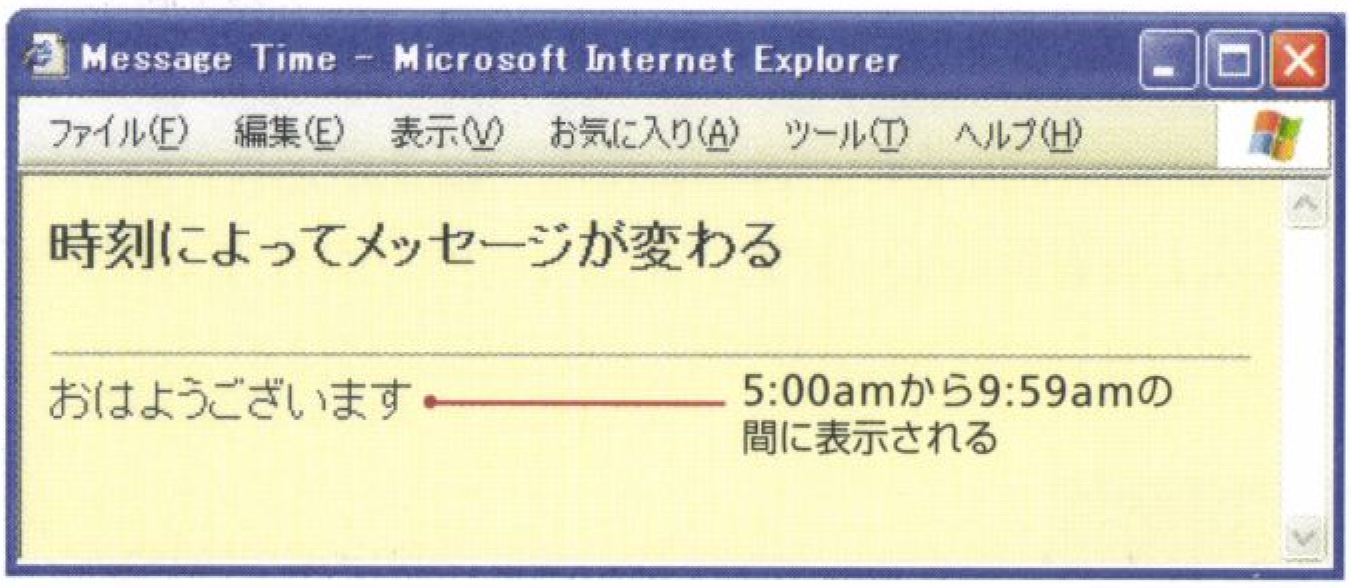
時間によって異なる挨拶文を表示する

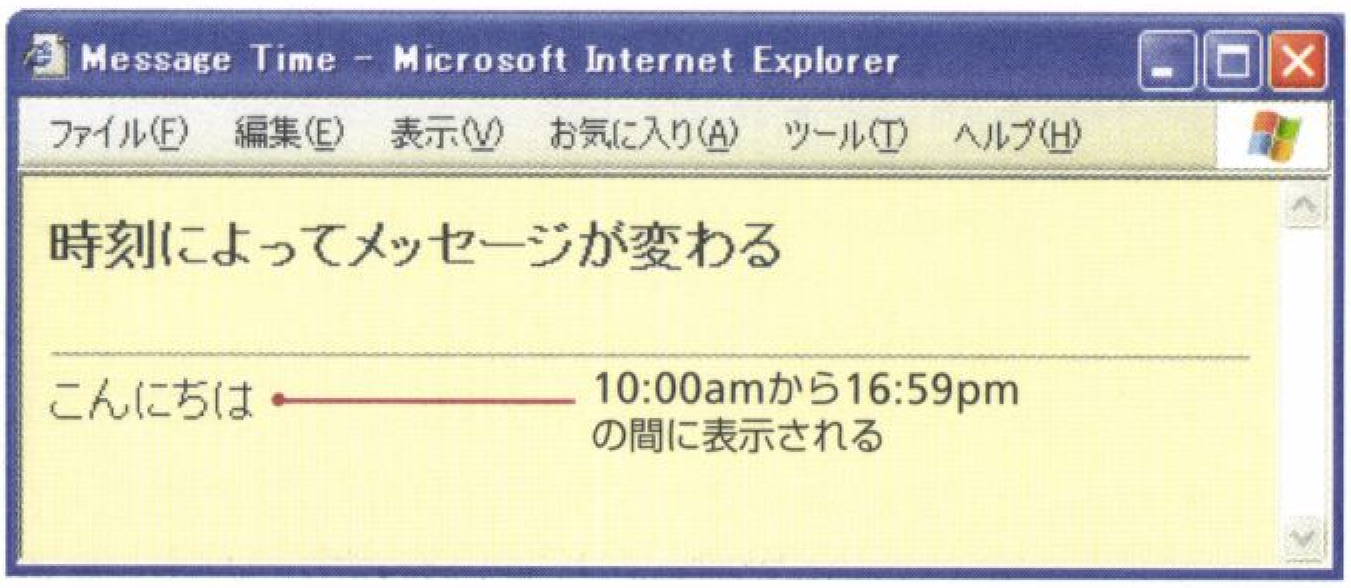
Dateオブジェクト名.getHours()

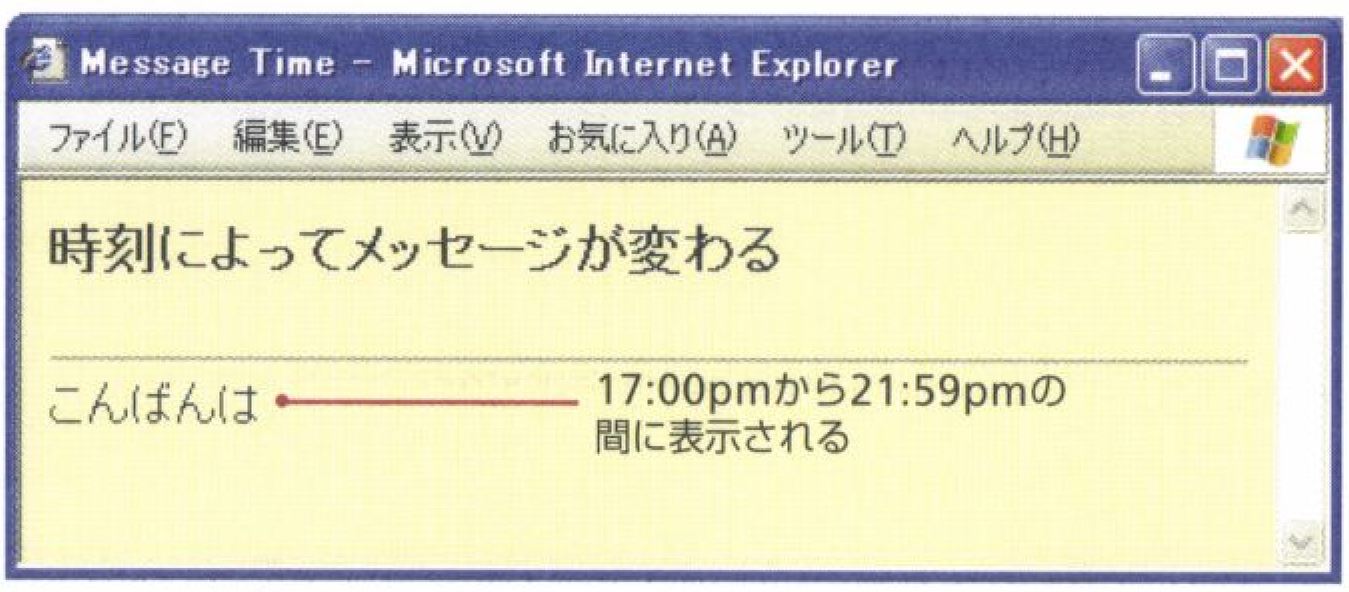
if～else文

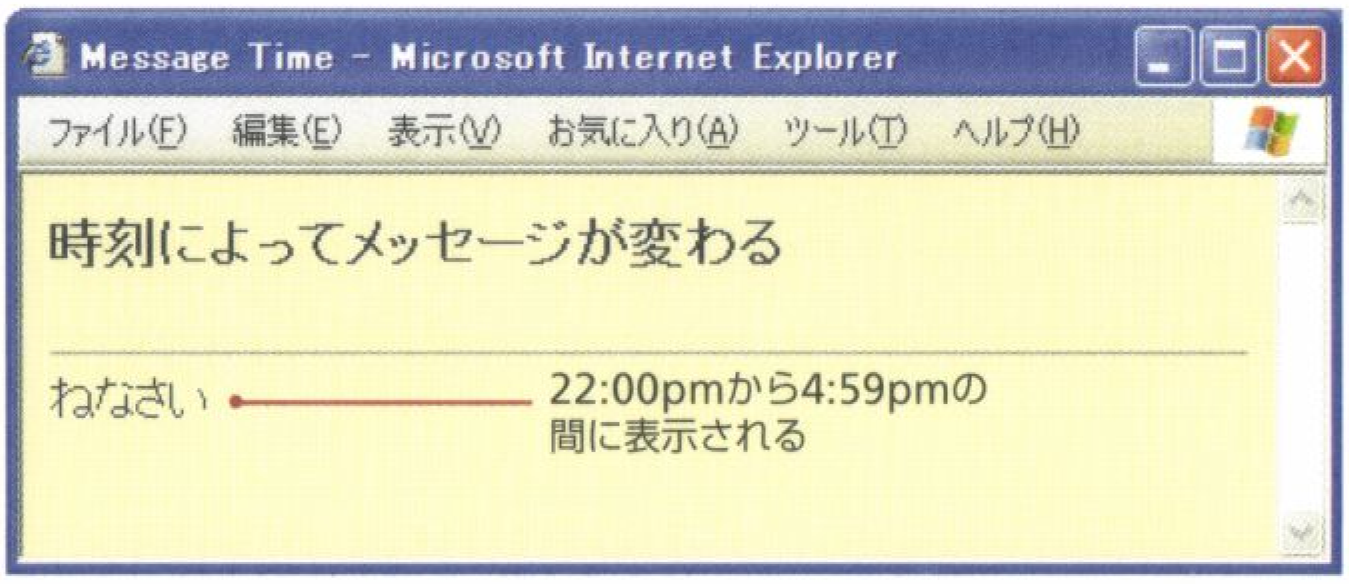
アクセスする時間によって、異なる挨拶文を表示するWebページを実現するスクリプトです。簡単なものですが、案外よい効果を生みます。ここではメッセージだけ変更しましたが、背景色やスタイルシートのテンプレートなどを変えることにより、朝と夜のWebページのイメージを変えるといったことも可能になります。

DISPLAY









SOURCE timemessage.html

```
<html>
<title>Message Time</title>
<script language="javascript">
<!--
function printMessage(){
   date=new Date(); ―――――――――――――――――――――①
   myHour=date.getHours(); ―――――――――――――――②
   if(5<=myHour && myHour <10){
      document.write("おはようございます");
   }
   else if( 10<=myHour && myHour <17 ){
      document.write("こんにちは");
   }
   else if( 17<=myHour && myHour <22){
      document.write("こんばんは");
   }
   else{
      document.write("ねなさい");
   }
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>時刻によってメッセージが変わる</h3>
<hr />
<script language="javascript">
```

```
<!--
   printMessage();
//-->
</script>
</body>
</html>
```

D E T A I L S

このスクリプトは非常に簡単なものです。まず①の部分で、現在時刻を取得するためにDateオブジェクトを生成します。

```
date=new Date();
```

その後、生成されたDateオブジェクトから「時」を取り出します(②の箇所)。

```
myHour=date.getHours();
```

「時」を取り出すにはgetHoursメソッドを用います。取り出した「時」は変数myHourに格納されます。

後は、if〜else文を用いて、時間により異なった処理を行っています。

```
if(5<= myHour && myHour <10)
```

上記のif文の条件は、5時から10時までの間ということです。「<10」ですから、10時になったら、この条件からは外れています。

この条件分岐を多くすることで、より細かな処理が可能となります。

カウントダウン

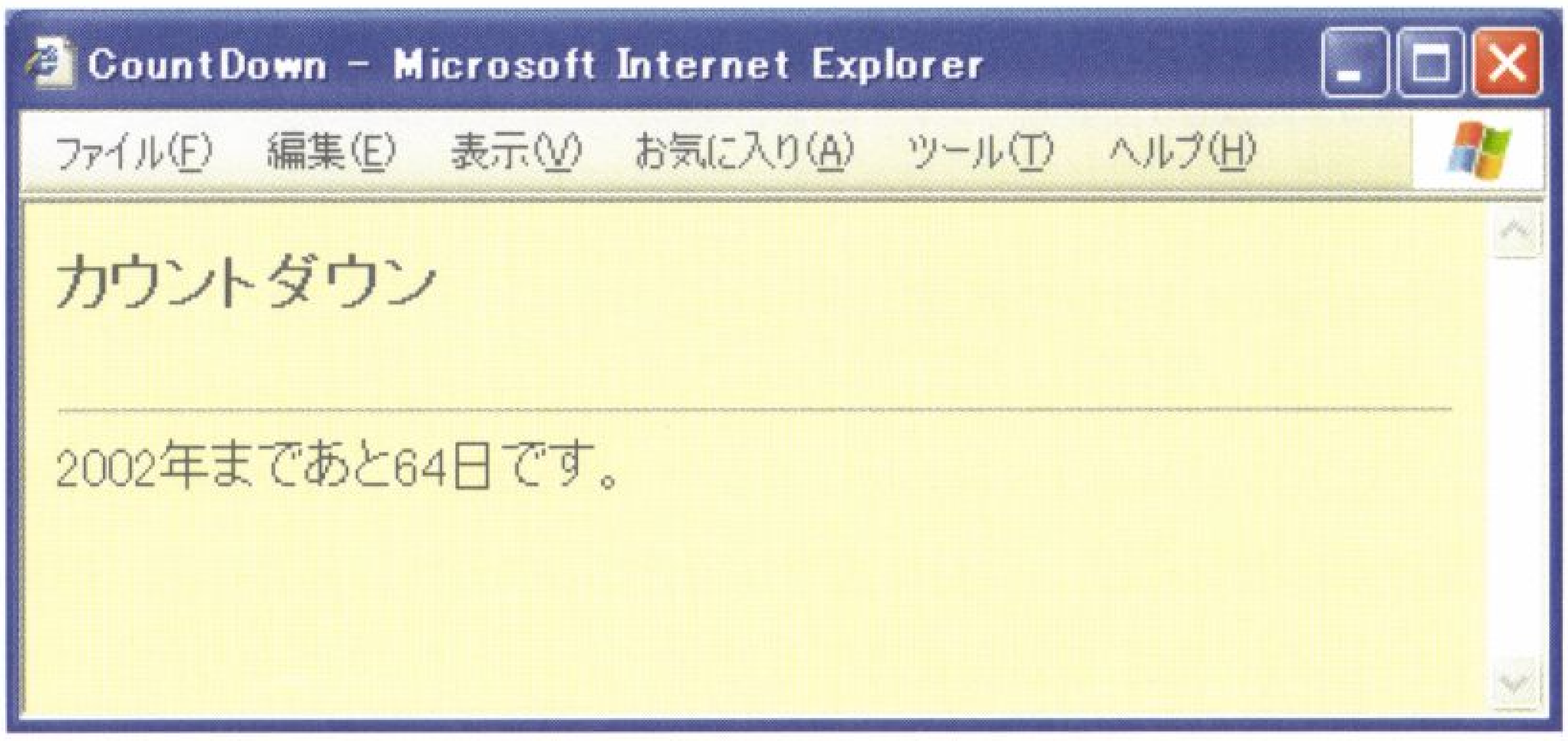
```
var 変数 = new Date(西暦, 月, 日);
```

```
Math.ceil( 値 )
```

```
Math.floor( 値 )
```

このスクリプトは、設定された日まで、後何日かを表示するものです。恋人の誕生日、結婚記念日、原稿の締め切りなどまでカウントダウンすることができます。

DISPLAY



SOURCE countdown.html

```
<html>
<title>CountDown</title>
<script language="javascript">
<!--
var SysDate=new Date();
var AniDate=new Date(2002,0,1);  ①
days=(AniDate.getTime()-SysDate.getTime())/(24*60*60*1000);  ②
//-->
</script>
<body bgcolor="#fff8dc">
<h3>カウントダウン</h3>
<hr />
<script language="javascript">
```

```
<!--
    document.write("2002年まであと",
➡       Math.ceil(days),"日です。");  ――③
//-->
</script>
</body>
</html>
```

DETAILS

このスクリプトでは、カウントダウンの対象として設定した日にちから、現在の日にちを引くことにより、後何日あるかを計算しています。日にちの設定にはDateオブジェクトを用います。設定をする書式は次のようになります。

```
var 変数 = new Date(西暦, 月, 日);
```

サンプルでは2002年1月1日をカウントダウンの対象としています(①の箇所)。このとき注意が必要なのが、「月」に設定する値は0からはじまるので、指定したい月よりも「1」少ない数値を設定することです。たとえば、1月なら「0」、12月で「11」を設定することになります。

この後、設定された指定日から現在の日付を引くのですが、ここでいくつか考えなくてはならない問題があります。

まず、引かれた結果はミリ秒(1/1000秒)単位で表現されるということです。この結果を、「日」に変換する必要があります。変換は、1日に相当する「24(時間)×60(分)×60(秒)×1000(1秒=1000ミリ秒)」で割り算します。この引き算と割り算の処理を行っているのが②の部分です。なお、もしも設定した日を過ぎている場合は、返ってくる値がマイナスになります。

次に、このままでは、割り算した結果が必ずしも整数にならないということです。たとえば、「あと10.38475日です」といわれてしまいます。これではあまりうれしくないので、整数にしなくてはなりません。この処理には切り上げを行うMathオブジェクトのceilメソッドを使っています(③の箇所)。なお、値の切り上げ、切り捨ては、次の書式で行うことができます。

```
Math.ceil( 値 )   ←―― 切り上げ
Math.floor( 値 )  ←―― 切り捨て
```

これで小数の部分が切り上げられて整数になり、指定した日までの日数を表示することができます

COLUMN

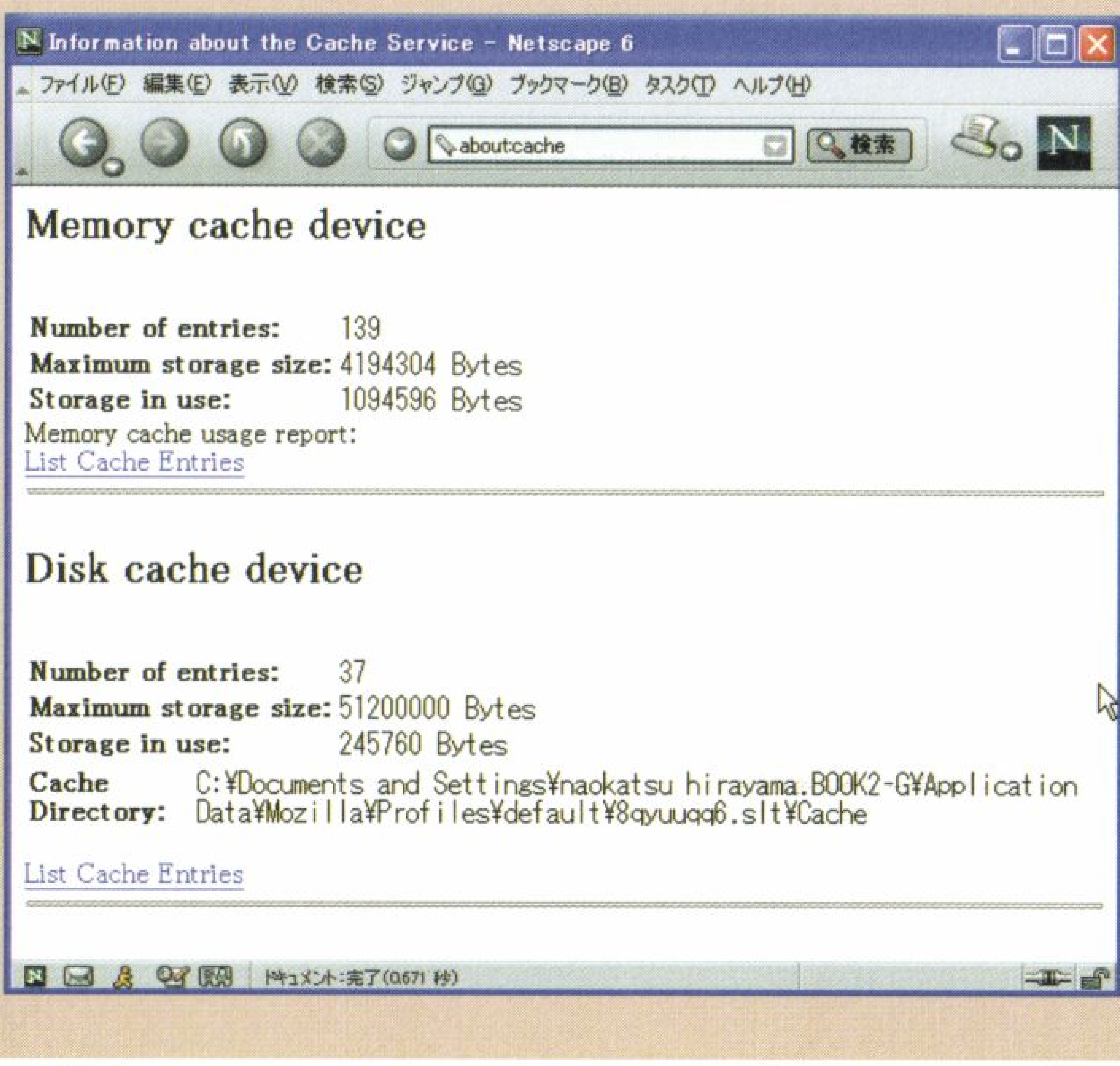
Netscape Navigatorの隠しコマンド

NNにはさまざまな隠しコマンドがあります。P.180で紹介している、内蔵デバッガの呼び出しもそのうちの1つです。

その他にも、NNについての情報を表示する「about:」、キャッシュファイルの内容を表示する「about:cache」などがあります。コマンドの入力には、「;」(セミコロン)ではなく「:」(コロン)を使うことに注意してください。

なお、NN6では、こういったコマンドが、NN4と比較すると減っているようです。

▼NN6で「about:cashe」を実行したところ



カレンダーを表示する

Dateオブジェクト名.setDate(指定する日付)

if(((西暦%4==0)&&(西暦%100!=0))||(西暦%400==0))

日付を表示するスクリプトを応用して、カレンダーを表示するスクリプトを作ることができます。最近では業務システムをイントラネットで構築していて、こういったカレンダーをJavaScriptで作るといったことはよく行われています。

DISPLAY



SOURCE showcalender.html

```
<html>
<title>Show Calendar</title>
<script language="JavaScript">
<!--
function showCalender(){
    var year, today, startDay;
    var date=new Date();
```

DATE・TIME

```
var monthdays=new Array
    (31,28,31,30,31,30,31,31,30,31,30,31);  ①
var days=new Array
    ("Sun","Mon","Tue","Wed","Thr","Fri","Sat");  ②
year=date.getFullYear();
today=date.getDate();

// 年、うるう年の設定
if(((year%4==0)&&(year%100!=0))||(year%400==0)){  ③
    monthdays[1]=29;
}
thisMonthdays=monthdays[date.getMonth()];  ④
date.setDate(1);                           ⑤
startDay=date.getDay();

// 年・月表示
document.write("<table border='1'>");
document.write("<tr><th colspan='7' bgcolor='fffafa'>");
document.write(year,'年',date.getMonth()+1,'月');
document.write("</th></tr>");

// 曜日の表示
document.write("<tr>");
for(i=0;i<7;i++){  ⑥
    if(i==0){
        document.write("<th bgcolor='#ffb6c1'>",
            days[i],"</th>");
    }
    else if(i==6){
        document.write("<th bgcolor='#6495ed'>",
            days[i],"</th>");
    }
    else{
        document.write("<th>",days[i],"</th>");
    }
}
document.write("</tr>")

// 日にちの表示
document.write("<tr>");
```


DATE・TIME

```
    var col=0;
    for(i=0;i<startDay;i++){
        document.write("<td> </td>");
        col++;
    }
    for(i=1;i<=thisMonthdays;i++){
        document.write("<td>");
        if(i==today){
            document.write("<font color='#ff0000'><b>");
        }
        document.write(i);
        if(i==today){
            document.write("</b></font>");
        }
        document.write("</td>");
        col++;
        if(col==7){
            document.write("</tr>¥n<tr>");          ⑦
            col=0;
        }
    }
    document.write("</table>");
}
//-->
</script>
<body  bgcolor="#fff8dc">
<h3>カレンダーを表示する</h3>
<hr />
<script language="javascript">
<!--
    showCalender();
//-->
</script>
</body>
</html>
```

DETAILS

このサンプルには、このセクションでこれまでに説明してきた、日付を扱うスクリプトのさまざまなテクニックが使われています。基本的には、Dateオブジェクトを使って現在の日付を取得し、それを加工してカレンダーを生成するのに必要な情報を生み出しています。

月の一覧を表示したカレンダーを作成するのに必要となるのは、「現在の月が何日まであるのか」ということと、「何曜日からはじまるのか」ということです。それぞれの月が何日まであるのかというのは、ごく一部の例外(うるう年)を除いて決まっています。何日あるかを設定しているのが①の部分です。Arrayオブジェクトを用いて配列(Monthdays配列)を作成し、1月から12月がそれぞれ何日あるかを設定しています。

また、②の部分では、同様に配列を用いて、曜日の一覧(Days配列)を作成しています。さらに、年と日付をそれぞれ取り出し、変数「year」、「today」に設定し4桁の西暦を取得するためにgetFullYearメソッドを用いています。

そして、③の部分で、月に何日あるかについての例外である、うるう年の処理を行っています。うるう年は4年に一度やってくるのですが、次のような条件で特定することができます。

- 西暦が4で割り切れて、100で割り切れない年はうるう年
- 上記の例外として400で割り切れる年はうるう年

判定した結果、うるう年に該当する年には、2月の日にち(monthdays[1])の数字を29日にしています。

うるう年の処理が終わったら、現在の月に何日あるかということを保持しておく変数「thismonthDays」を設定します。④の部分で、monthdays配列の添え字にdate.getMonthと指定して月の情報を設定し、今月は何日あるかを取り出しています。

この後、⑤の部分で、今月が何曜日からはじまるかを調べます。まずDateオブジェクト「date」の日付を、setDateメソッドを使って「1日」に設定します。現在の日付は、すでに変数todayにとってあるので問題ありません。次に、この1日にセットしたdateの曜日を取り出し、「startDay」に設定しています。これで、現在の月が何曜日からはじまったかを取得することができることになります。



後は、表示するための記述を行います。最初に西暦と月を、次に⑥の箇所で、for文を使ったループを使って曜日を表示します。

その後、startDayの曜日までを空白にするために、for文を使ったループで、startDayの保持している数値まで空のセル（<td> </td>）を挿入します。

ここまでが終わったら、実際に日付を表示していきます。1日の1からthisMonthDaysに設定されている数値まで表示されます。

また、横方向に数値（または空白）がいくつ並んだかを数える変数colが7回カウントしたら行を替えて、新しい行に日付を表示していきます（⑦の部分）。表示が終わったら最後にテーブルを閉じます。

9-1 WIN e 4↑ N 4↑ MAC e 4↑ N 4↑

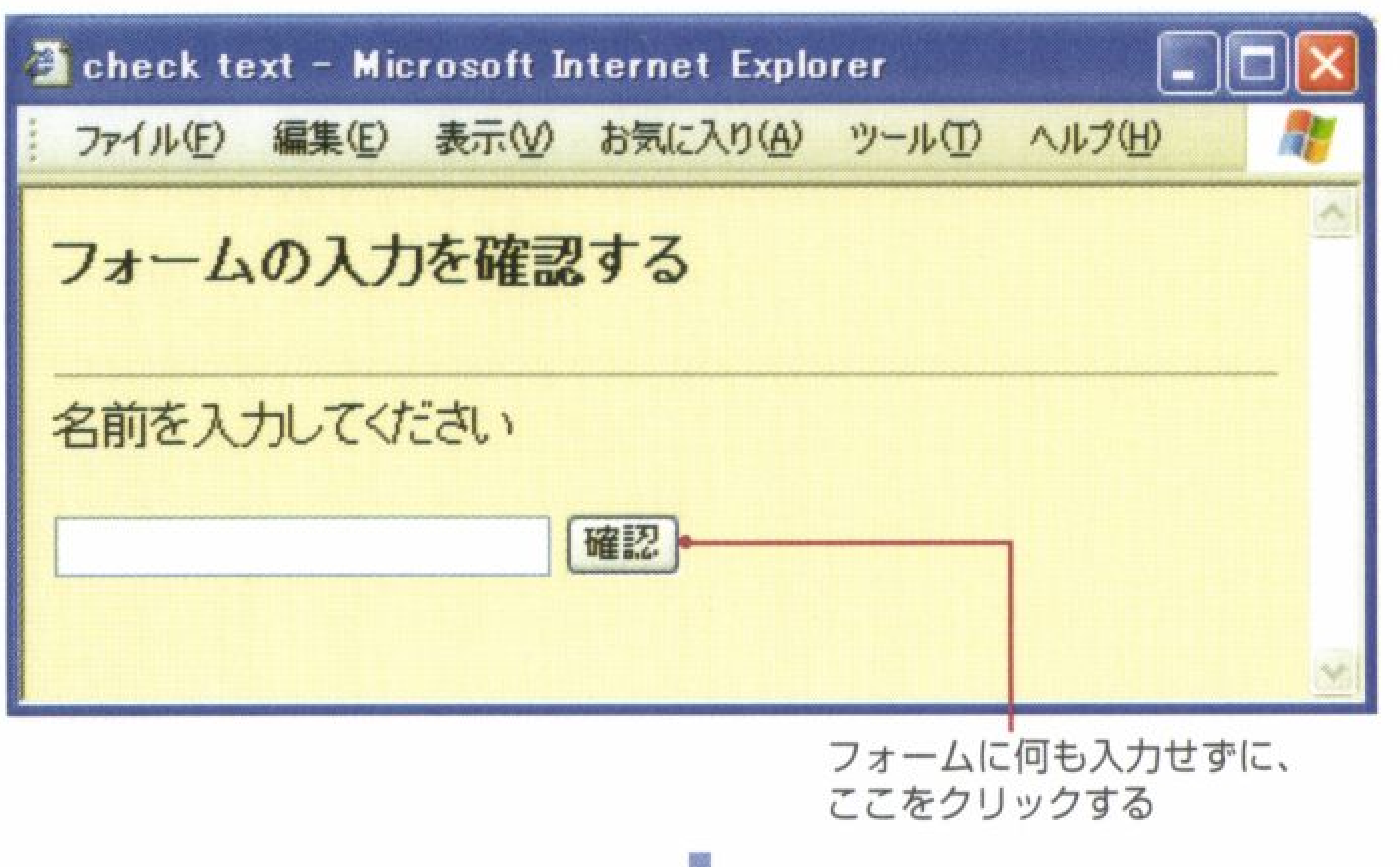
項目が入力されているかチェックを行う

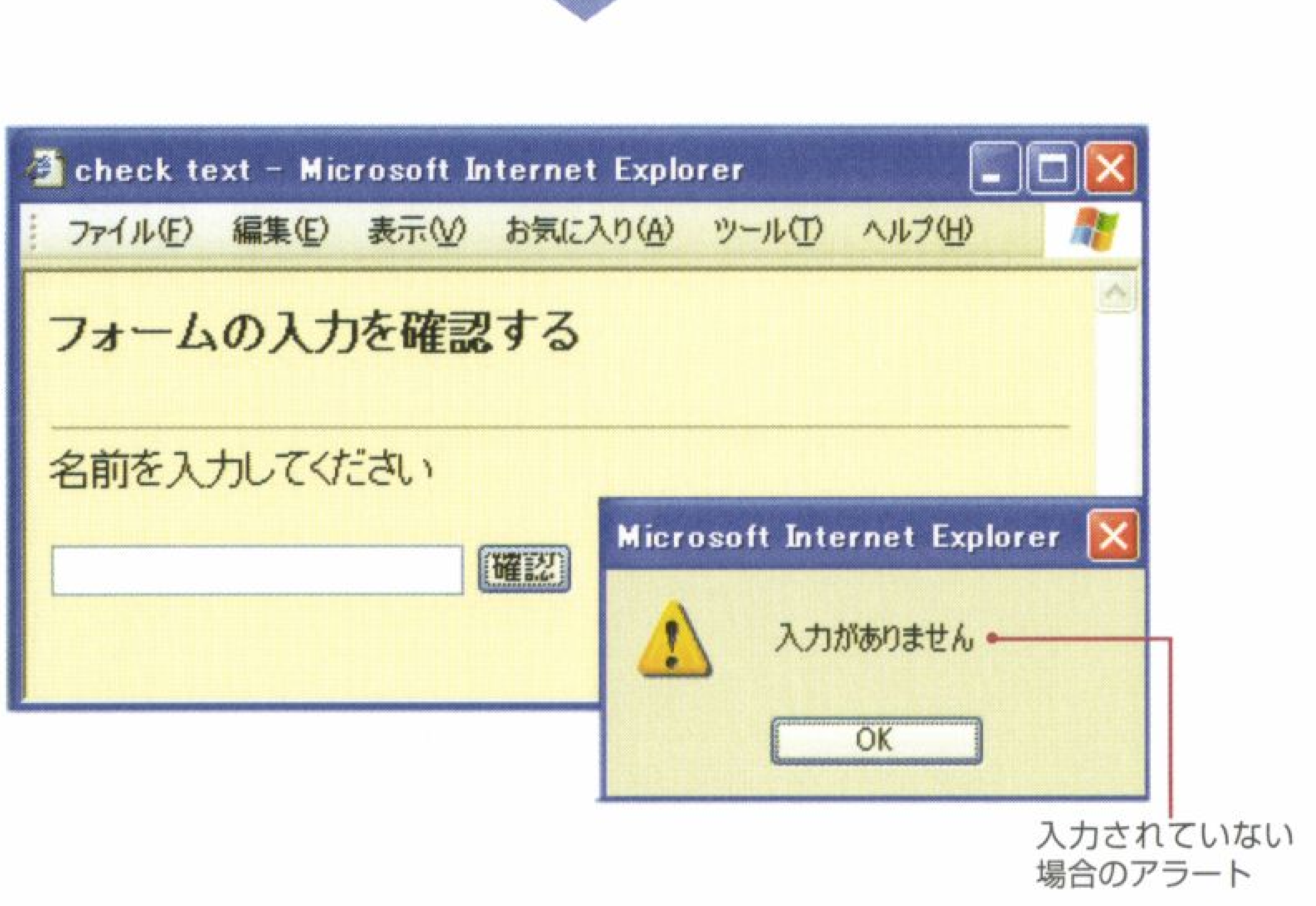
document.フォーム名.フォーム要素名.value

インタラクティブなWebページや、電子商取引のWebページではさまざまなフォームが使われます。このフォームは、ユーザーが自由に入力できたり、選択できるようになっているので、ユーザーの入力ミス、または入力されないといったことがよくあります。こんな場合に、入力データがサーバー側に送られてから、サーバーサイドでチェックするシステムでは、パフォーマンスが悪くなってしまいます。そこで、フォームに入力がされているかどうかを、クライアント側で確認できるようにしてみます。

DISPLAY

▼何も入力していない場合のアラート





SUPPORT

▼入力がある場合のアラート



SOURCE

checktext.html

```
<html>
<title>check text</title>
<script language="javascript">
<!--
function checkText(){
    if(document.myForm.myText.value==""){        ①
        alert("入力がありません");
    }
    else{
        alert("入力があります");
    }
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>フォームの入力を確認する</h3>
名前を入力してください<br/>
<hr />
<form name="myForm">
    <input name="myText" type="text" size="30" />
    <input type="button" value="確認" onclick="checkText()" />
```

▼

SUPPORT

```
</form>
</body>
</html>
```

DETAILS

このサンプルでは、Webページ上に配置されたテキストフォームに入力があるかどうかを確認します。

ユーザーが［確認］ボタンをクリックすると、onclickイベントハンドラで指定されたcheckText関数が呼び出されます。checkText関数ではテキストフォームに入力があるかどうかを調べます。このとき、テキストフォームの値にアクセスするのに、次のように指定しています(①の箇所)。

```
document.myForm.myText.value
```

この記述は、このWebページ(document)の、myFormと名付けられたフォームの、myTextと名付けられたフォームの要素の値(value)という意味です。これはフォームや、フォームの要素に、name属性を用いて名前が付けられているのでこのように指定することができます。もしも、名前が付いていない場合には、次のように指定することができます。

```
document.forms[0].elements[0].value
```

これは、Webページ(document)内の、最初のフォーム(forms[0])の、最初の要素(elements[0])を表します。

このようにして、checkText関数でアクセスした値が、空「""」でないかどうかを確認しています。

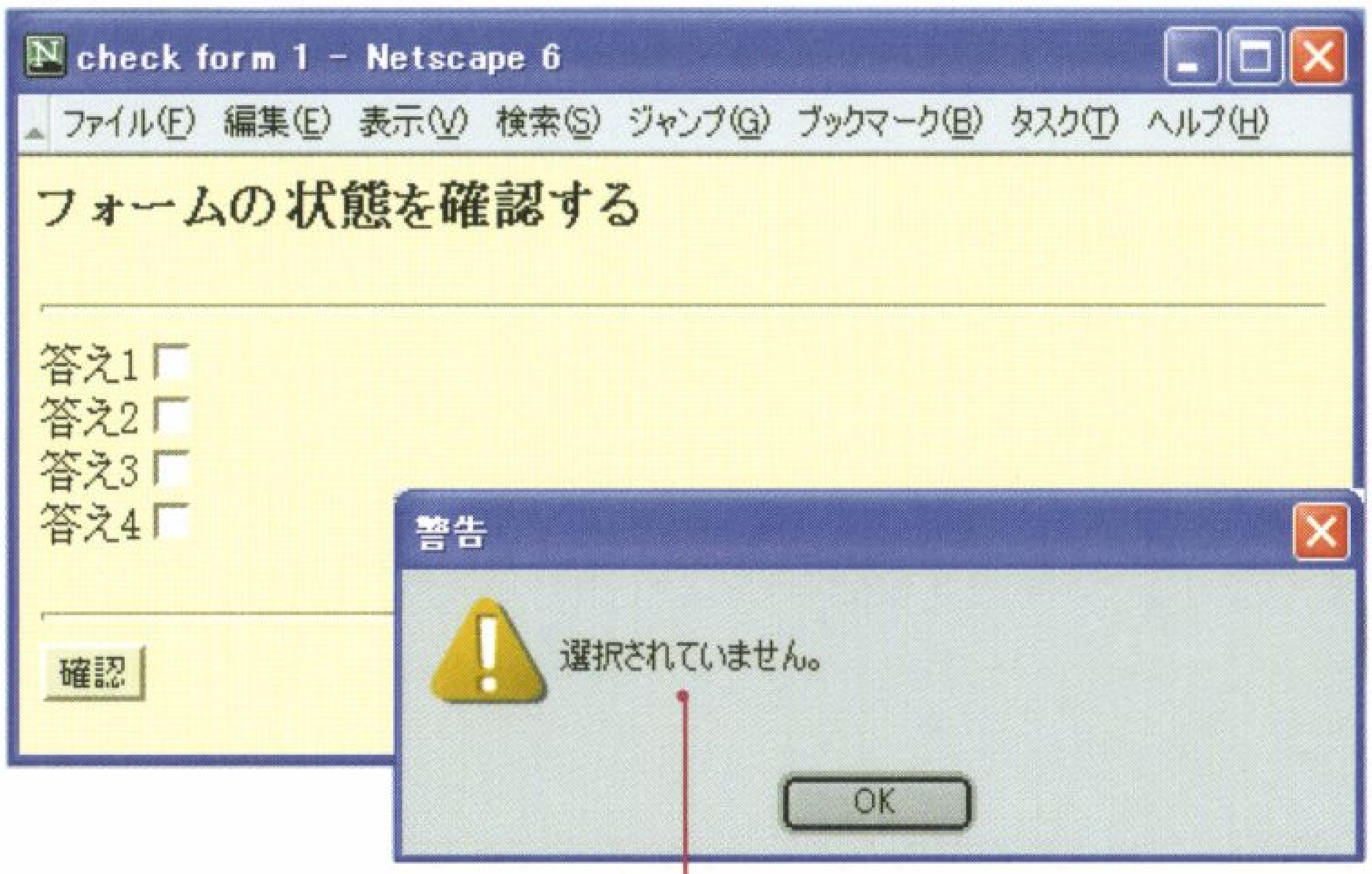
WIN 4↑ 4↑ MAC 4↑ 4↑

チェックボックスフォームの状態を確認する

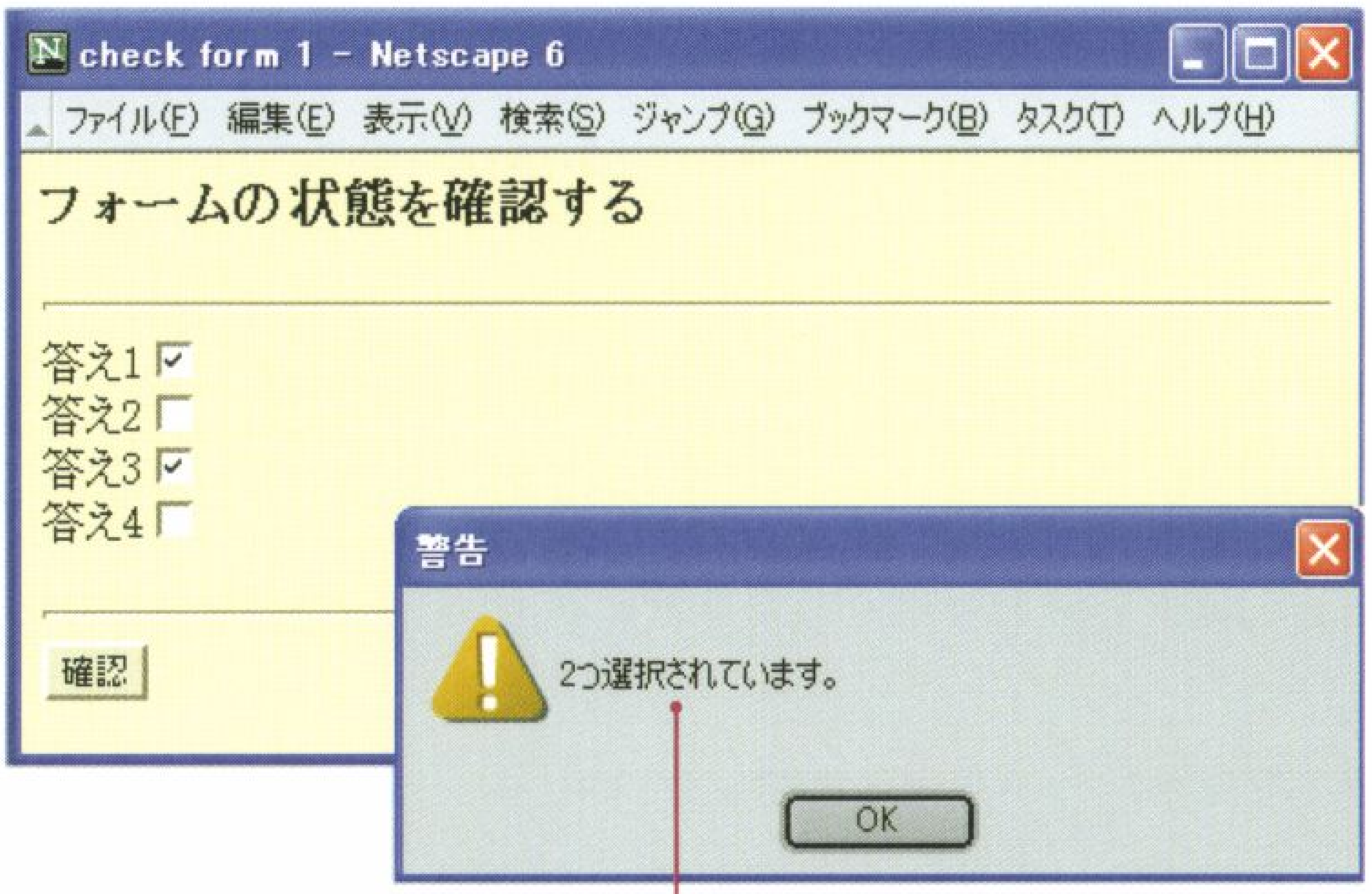
document.フォーム名.チェックボックス名.length

フォームの中でも、複数の選択肢を選択させることができる、チェックボックスフォームの入力状態を確認するスクリプトです。いくつのチェックボックスにチェックされ、選択されているかを確認します。

DISPLAY



チェックされていないときのアラート



2つチェックされているときのアラート

SOURCE

checkform1_html

```
<html>
<title>check form 1</title>
<script language="javascript">
<!--
function checkForm(){
   var i, count=0;
   for(i=0; i<document.myForm.rForm.length ; i++ ){
      if(document.myForm.rForm[i].checked==true){   ①
         count++;
      }
   }
   if(count>0){   ②
      alert(count + "つ選択されています。");   ③
   }
   else{
      alert("選択されていません。");
   }
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>フォームの状態を確認する</h3>
<hr />
<form name="myForm">
   答え1<input name="rForm" type="checkbox" value="answer1" />
   <br />
   答え2<input name="rForm" type="checkbox" value="answer2" />
   <br />
   答え3<input name="rForm" type="checkbox" value="answer3" />
   <br />
   答え4<input name="rForm" type="checkbox" value="answer4" />
   <br />
</form>
<hr />
<form>
   <input type="button" value="確認" onclick="checkForm()" />
</form>
```

```
</body>
</html>
```

DETAILS

このサンプルでは、Webページ上に配置されている4つのチェックボックスが入力されているかどうかを確認しています。

確認 ボタンをクリックすると、onclickイベントハンドラに指定されたcheckForm関数が呼び出されます。checkForm関数では、フォーム内のrFormと名付けられたチェックボックスが、いくつチェックされたかを確認して数を返します。チェックボックスの数を確認するために、lengthプロパティを用いて、次のように記述しています。

```
document.myForm.rForm.length
```

上記の記述は、Webページ(document)の、myFormと名付けられたフォームの、rFormと名付けられたチェックボックスの数(length)という意味です。この数値を用いてfor文でループし、その中でチェックボックスを個別に確認していきます。ループは、チェックボックスの個数と同じ回数だけ実行されます。

チェックボックスの状態を調べるにはcheckedプロパティを用います。checkedプロパティは、選択されている場合には「true」が、選択されていない場合には「false」が返されます。そのため、①の部分で「true」の場合だけ、変数countに「1」を足していきます。

ループが終了したら、②の箇所で変数countを確認します。0より大きければチェックボックスのどこかにチェックがされているということなので、③の部分でアラートを表示します。そうでない場合はチェックがされていないので「選択されていません。」という表示を行います。

実際にこのサンプルを用いる場合には、アラートの表示を行っている部分で何らかの処理を組み込めばよいでしょう。

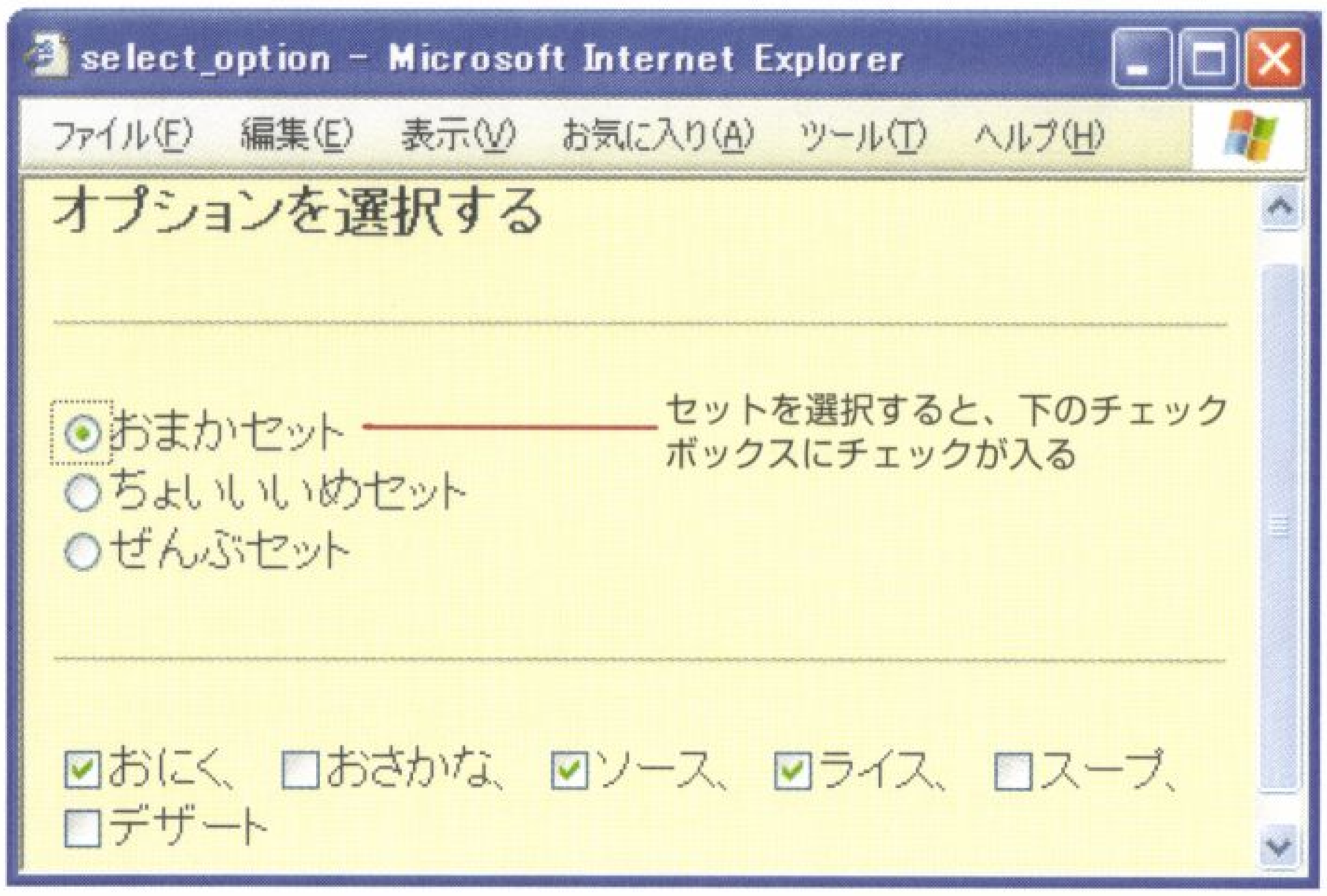
9-3 WIN MAC

オプションを選択する

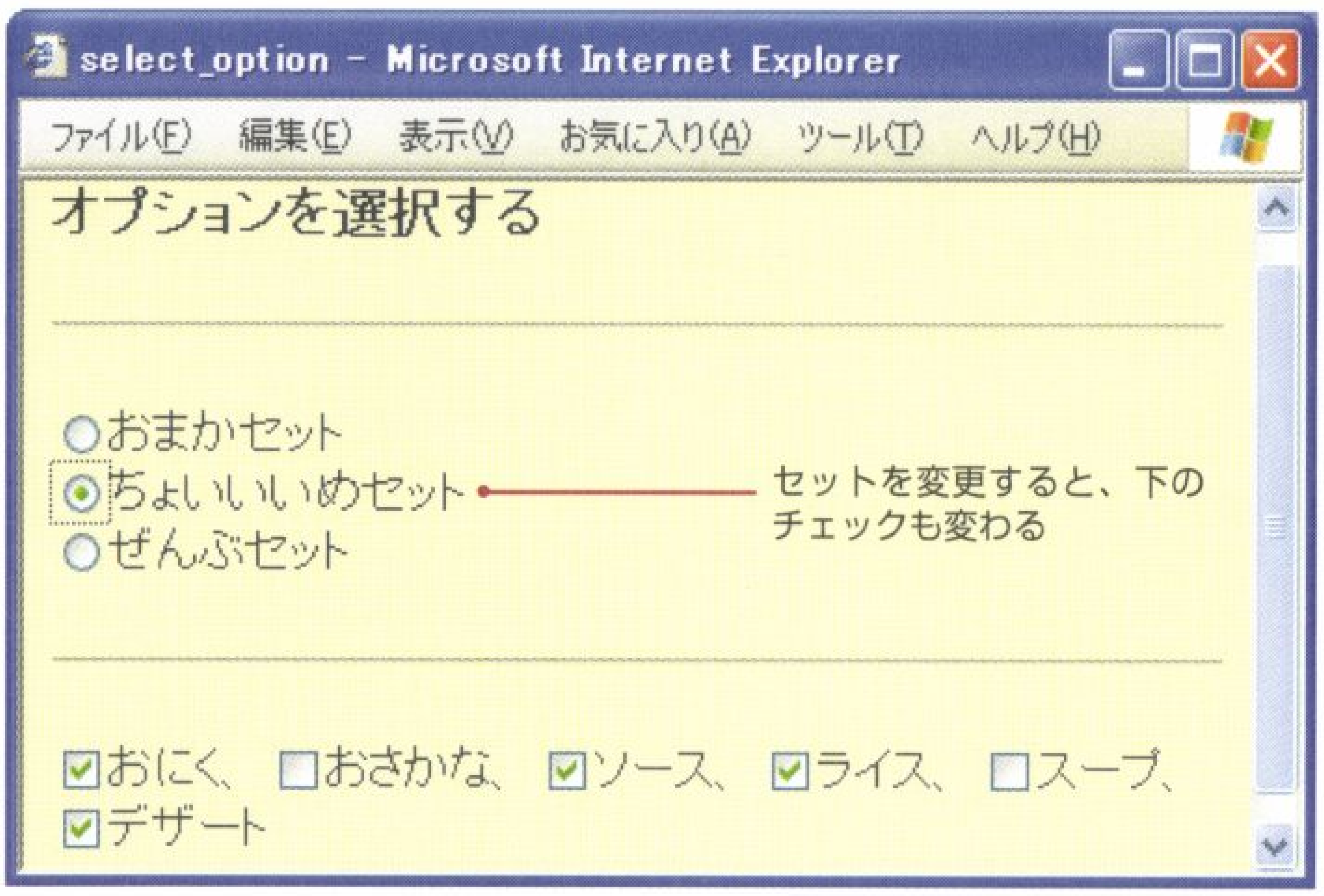
```
document.optionTarget.elements[i].checked =
true | false
```

インターネットでさまざまなものが購入できるようになってきました。しかし、Webページで商品が選択しにくかったり、操作が面倒だったりするものは敬遠されがちです。そこでセットを組んで、選択を容易にしている場合があります。そんな機能をスクリプトで実現するための基礎部分を解説します。

DISPLAY



↓



SOURCE

selectform.html

```
<html>
<title>select_option</title>
<script language="javascript">
<!--
var i, count, option_total;
var oarray = new Array();
function checkOption(charSelect){
    oarray = charSelect.split(",");  ②
    count = oarray.length;
    option_total = document.optionTarget.length;
    for(i=0;i<option_total;i++){  ③
➡       document.optionTarget.elements[i].checked=false;
    }
    for(i=0;i<count;i++){
➡       document.optionTarget.elements
➡       [oarray[i]].checked = true;  ④
    }
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>オプションを選択する</h3>
<hr />
<form name="myForm">
    <input type="radio" name="select1" value="v1"
➡       onclick="checkOption('0,2,3')" />おまかセット
    <br />
    <input type="radio" name="select1" value="v2"
➡       onclick="checkOption('0,2,3,5')" />ちょいいいめセット
    <br />
    <input type="radio" name="select1" value="v3"
➡       onclick="checkOption('0,1,2,3,4,5')" />ぜんぶセット
    <br />
</form>
<hr />
<form name="optionTarget">
<input nane="op0" type="checkbox">おにく、
```

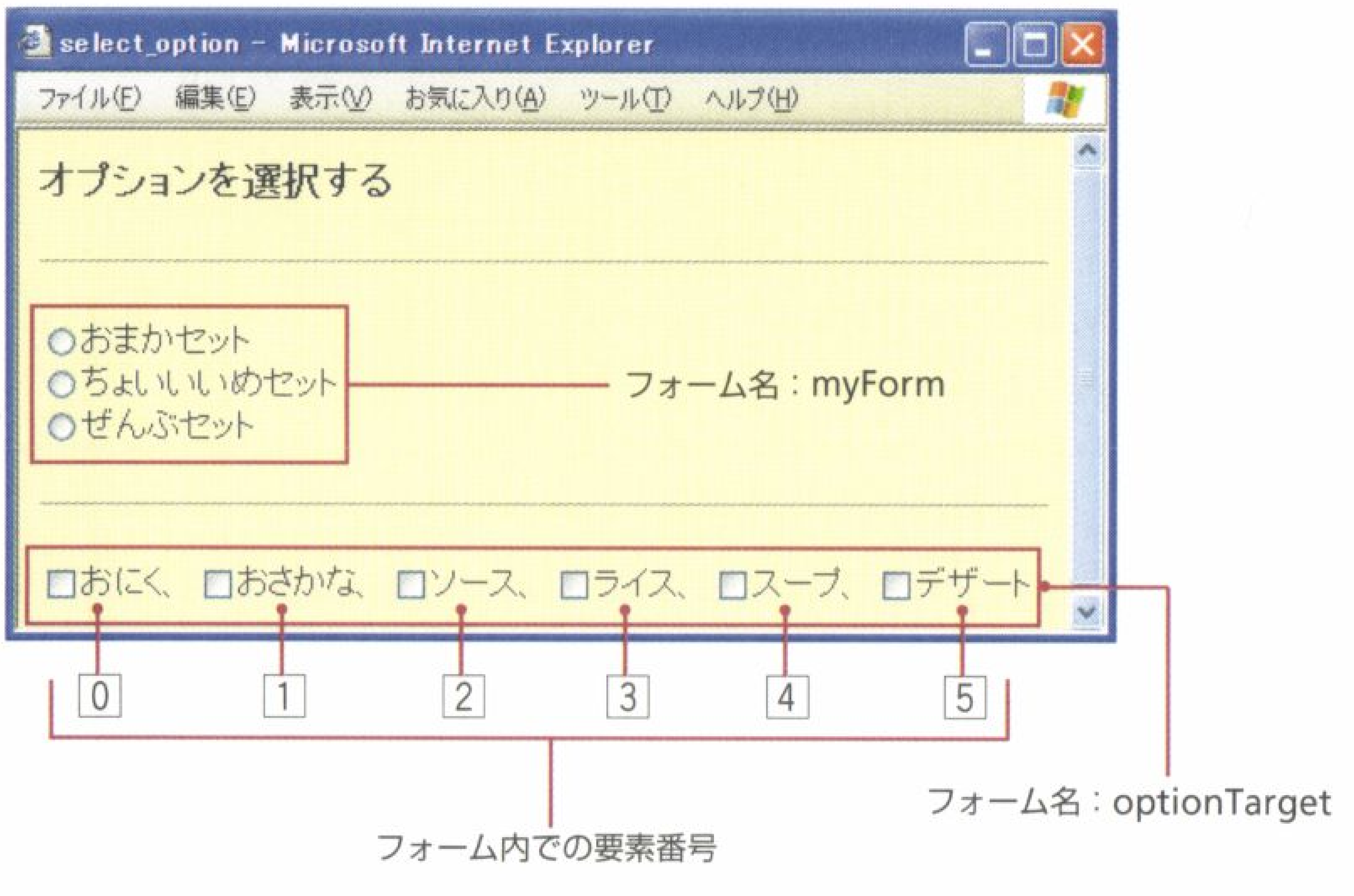
（①：3つの input 要素〜ぜんぶセットまで）

SUPPORT

```
<input nane="op1" type="checkbox">おさかな、
<input nane="op2" type="checkbox">ソース、
<input nane="op3" type="checkbox">ライス、
<input nane="op4" type="checkbox">スープ、
<input nane="op5" type="checkbox">デザート<br />
</form>
</body>
</html>
```

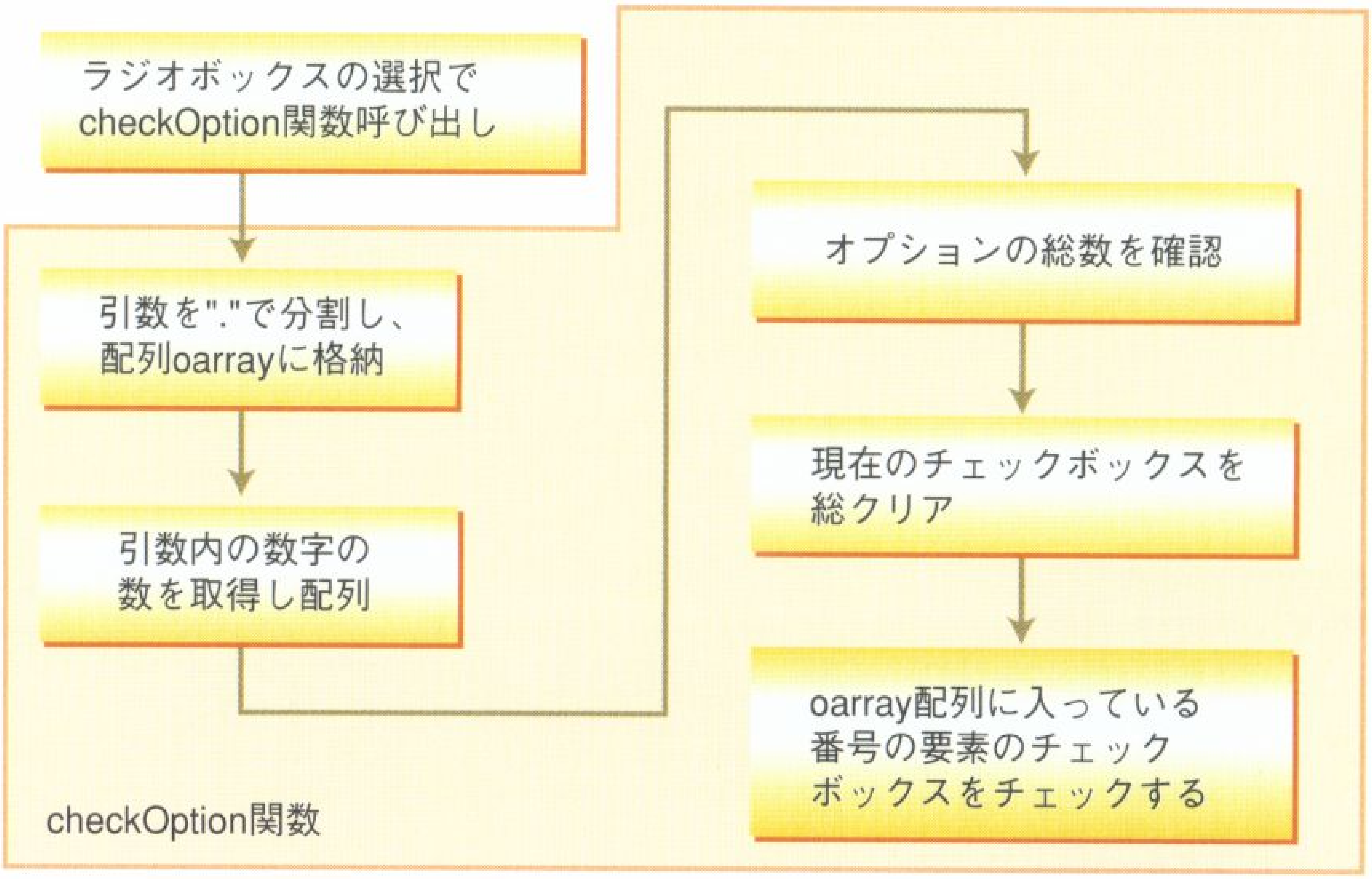
DETAILS

このサンプルは、メニューとして用意された択一選択フォームを指定すると、下に用意されている項目のどれが含まれているかをチェックボックスで表してくれるというものです。Webページ上にはフォームが2つ配置されています。最初のフォームは択一選択フォーム、2つ目のフォームはチェックボックスです。



動作としては、1つ目のフォームの選択肢をクリックすると、下のチェックボックスにチェックが行われるというものです。このチェックの部分はcheckOption関数に実装されています。

この関数の呼び出しは、1つ目のフォーム内の各ラジオボタンに対して設定されているonclickイベントハンドラで行います（①の箇所）。checkOption関数は引数として、optionTargetフォームのどの要素をチェックするかという番号をカンマ区切りしたものを取ります。



checkOption関数では、まず引数として受け取った数列(変数charSelectに格納されている)を分解する処理を行います。splitメソッドを用いて、「,」を分割記号として処理を行い、分解した結果は配列oarrayで受け取ります(②の箇所)。この後、配列oarrayの要素の数と、optionTargetフォーム内の要素の数を、lengthプロパティを用いて取得します。

取得したoptionTargetフォーム内の要素の数を用いて、フォーム内のチェックボックスフォームのチェックをすべてクリアします(③の箇所)。

最後に、配列oarrayに入っている数に対応するoptionTargetフォームの要素にチェックを付けていきます。このとき、配列oarrayの要素の中にはチェックを付けたいoptionTargetフォームの要素の番号が入っているので、④の箇所のような記述になります。この部分で注目するのはoarray[i]です。

```
document.optionTarget.elements[oarray[i]].checked
```

チェックしたい要素の番号に変わる

このように配列とfor文をうまく使うことで、簡便に記述できます。

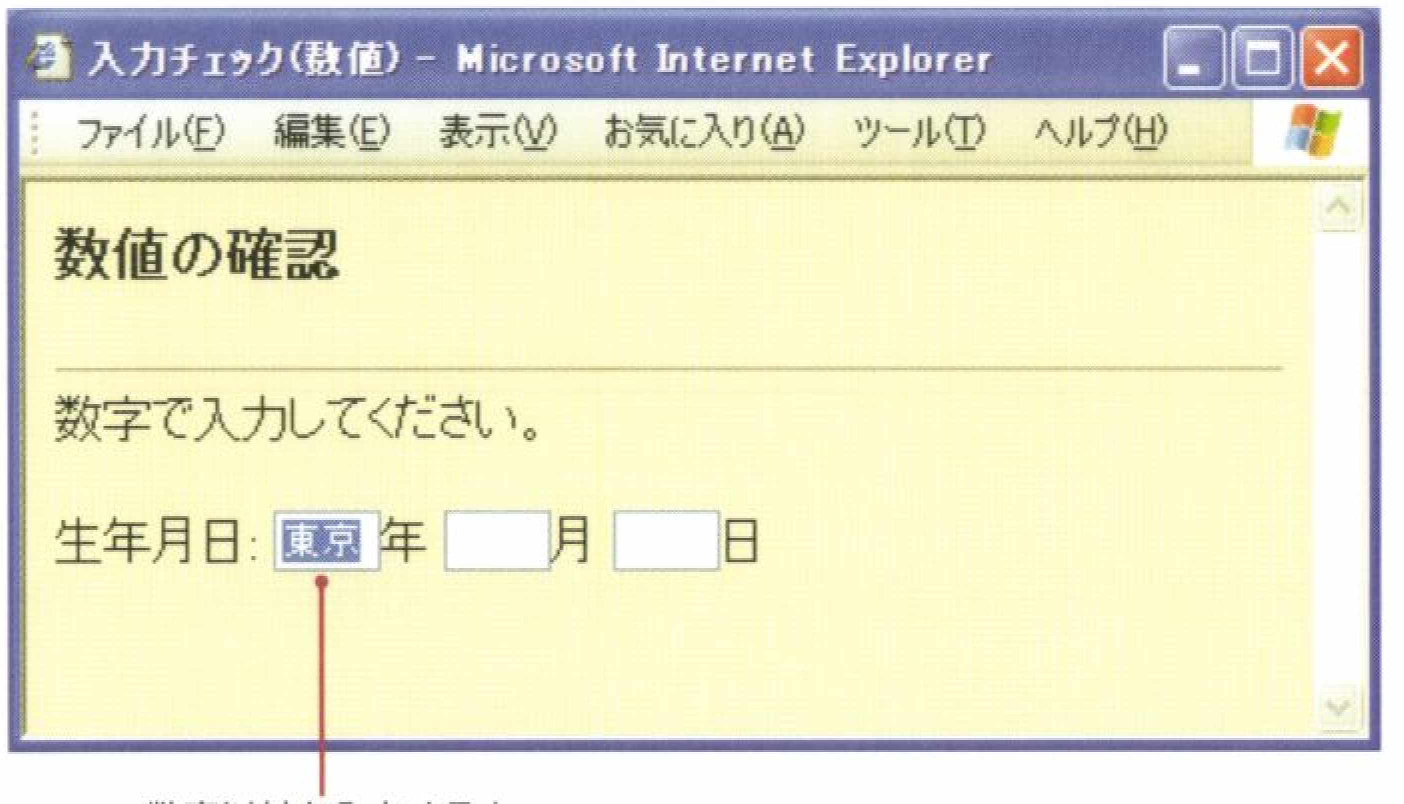
9-4 WIN e4↑ N4↑ MAC e4↑ N4↑

文字列を確認する1（数字）

isNaN(値)

これはフォームで入力された値が、数値かどうかチェックするスクリプトです。CGIなどでよく年齢や年月日を入力する場合があります。そんなときに入力された値が数値かどうかを、サーバーへ送信する前にチェックできれば、トラフィックを減らせるとともに反応をよくすることができます。

DISPLAY



数字以外を入力すると・・・



アラートが表示される

SUPPORT

SOURCE checknumber.html

```
<html>
<title>入力チェック(数値)</title>
<script language="javascript">
<!--
function isNumeric(ivalue){ ——①
   if(isNaN(ivalue))
      alert("数値ではありません。");
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>数値の確認</h3>
<hr />
数字で入力してください。<br />
<form>
   生年月日:
   <input type="text" onchange="isNumeric(this.value)"
➡      size="4" />年
   <input type="text" onchange="isNumeric(this.value)"
➡      size="4" />月
   <input type="text" onchange="isNumeric(this.value)"
➡      size="4" />日
</from>
</body>
</html>
```
(② marks the three <input> elements)

DETAILS

値が数値かどうかをチェックするにはisNaN関数を使います。書式としては以下のようになります。

```
isNaN( 値 )
```

isNaN関数は、値が文字列だと「true」、数字だけだと「false」を返します。数字かどうかを調べるならば、値が数字のときに「true」を返すのが普通かと思われ

ますが、これは逆になっています。NaNとは「Not a Number」(数字でない)ということです。

なぜこのような仕様になっているかというと、数字かどうかを調べるということは、つまり数字ならば何もしないでよいということだからです。そのためif文などと組み合わせて使うときには、このように逆になっているほうが都合がよいのです。

今回のスクリプトでは、数字かどうかを調べる部分を関数として使っています。こうすれば、複数の入力項目があってもそれぞれの場所にチェックするコードを実装しなくてすみ、コードを簡単にすることができます。

今回作成したisNumeric関数では、引数を変数「ivalue」で受け取り、それが数値でないならばアラートを出すようにしています(①の箇所)。また、②の部分のフォームの入力部には、<input>タグのonchangeイベントハンドラにより、値が入力されたときにisNumeric関数を呼び出すようにしています。そして、isNumeric関数へ渡す値は、次のような記述になっています。

```
this.value
```

これはthisオブジェクトの値を渡すという意味ですが、thisオブジェクトは自分自身、要するに<input>要素そのものを指しています。これでそれぞれの入力部に値が入力されたときにisNumeric関数が呼び出され、入力された値のチェックを行うことができます。

SUPPORT

9-5 WIN 4↑ 4↑ MAC 4↑ 4↑

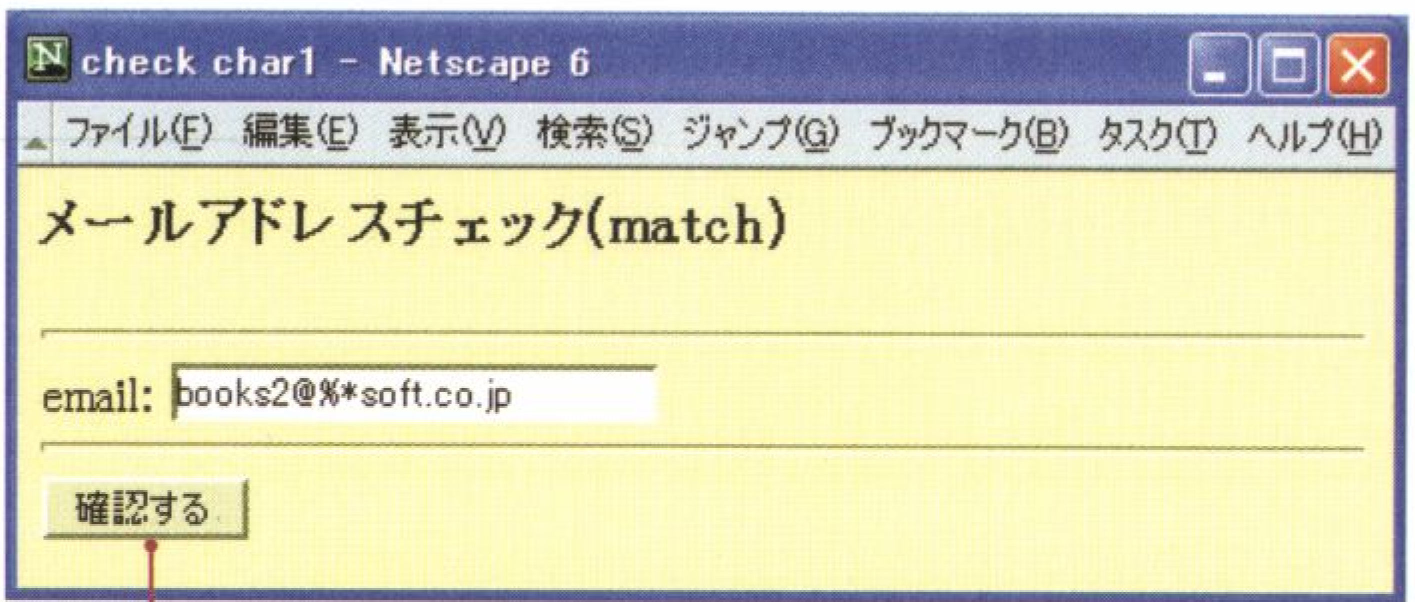
文字列を確認する2（特殊記号）

Stringオブジェクト.match(/[検索文字]/)

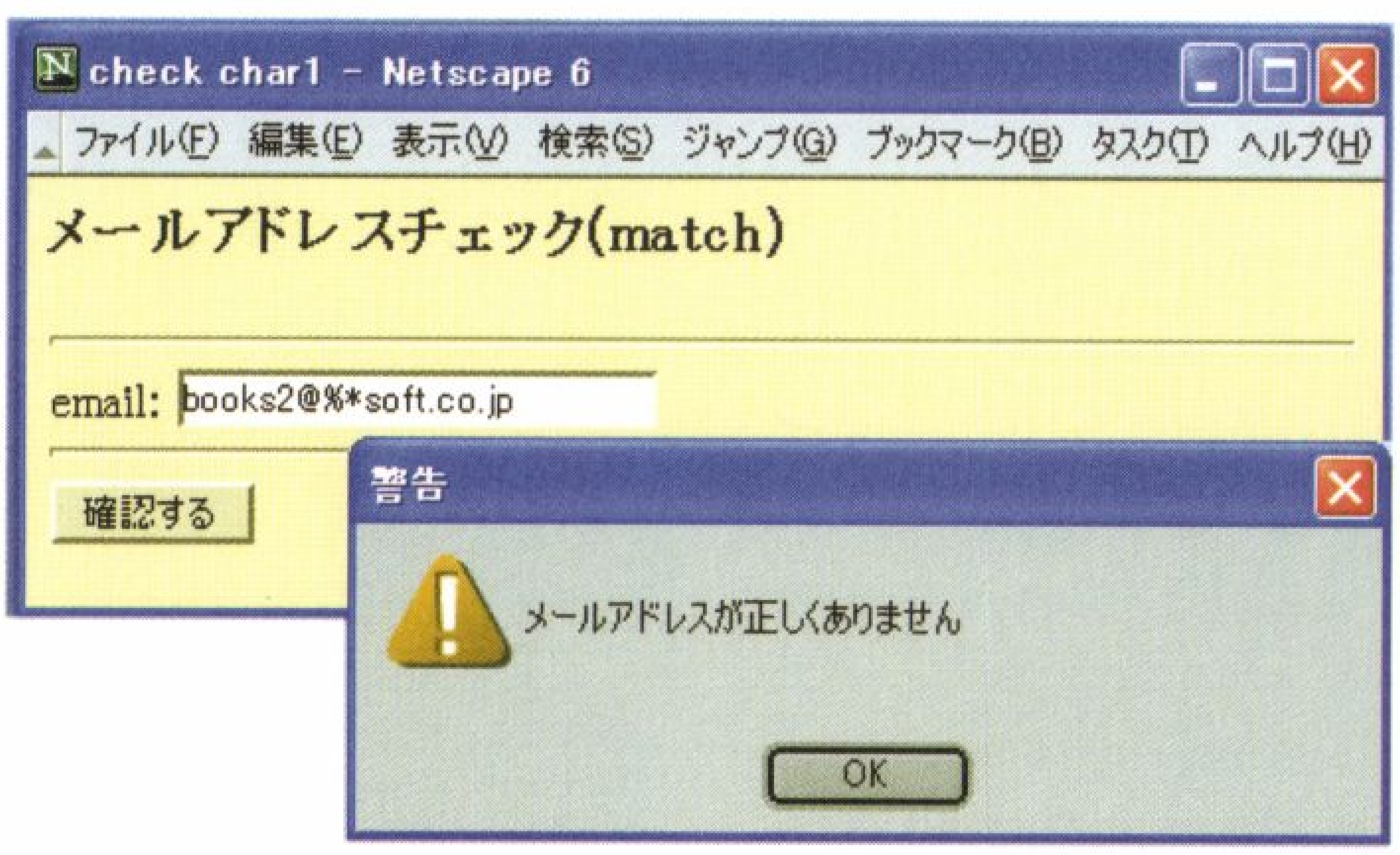
ユーザーに入力を促すWebページが結構多いと思いませんか？　たとえば、インターネット上で買い物をしようと思ったら、多くの情報を入力しなくてはなりません。そんなWebページを作る側にとっては、入力してもらった情報が間違っていては困ってしまいます。こういったことを防ぐには、入力された文字をチェックする処理を行います。

ここでは、特殊記号が使われているかどうかを、メールアドレスに不適切な文字列が入力されたかどうかを判別する例で解説してみます。

D I S P L A Y



特殊記号の入った間違ったメールアドレスを入力して、ここをクリックすると・・・



SOURCE checkchar1.html

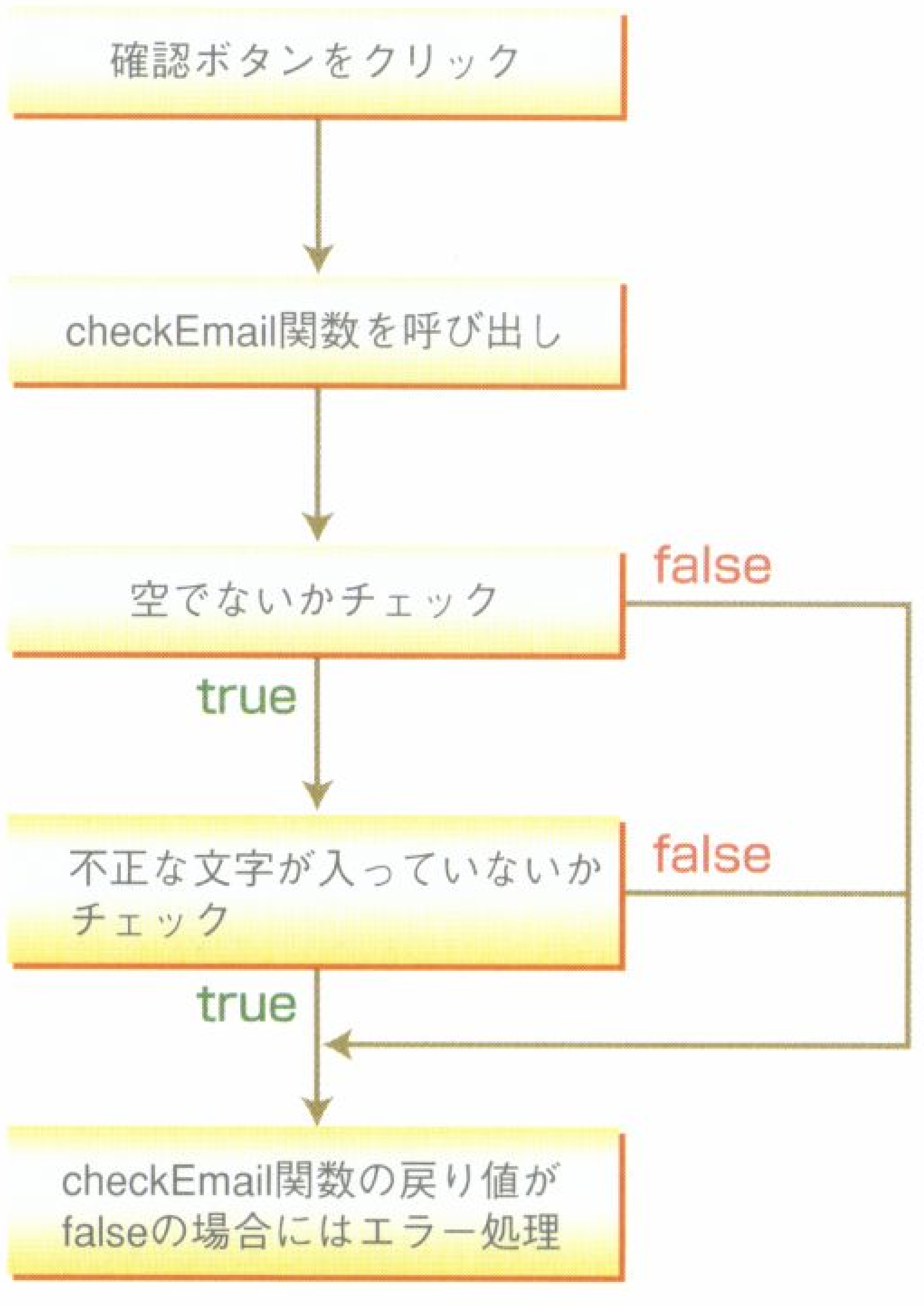
```
<html>
<title>check char1</title>
<script language="javascript">
<!--
function chackAll(){
   if(checkEmail(document.myForm.email.value)==false){   ——②
      alert("メールアドレスが正しくありません");
   }
}

function checkEmail(strTarget){
   if( strTarget.length == 0 ){   ——③
      return false;
   }
   if(strTarget.match
➡     (/[!"#$%&'¥(¥)=¥¥|,:;^¥[¥]¥{¥}`*?+><]/)){   ——④
      return false;
   }
   else{
      return true;
   }
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>メールアドレスチェック(match)</h3>
<hr />
<form name="myForm">
email: <input type="text" name="email" size="30" /><br />
<hr />
<input type="button" value="確認する"
➡  onclick="chackAll()" />   ——①
</form>
</body>
</html>
```

DETAILS

このサンプルでは、入力されたメールアドレスに、メールアドレスとして不適切な文字が入っていないかどうか確認するものです。通常メールアドレスには、半角英数文字と「@」（アットマーク）、「-」（ハイフン）、「_」（アンダースコア）、「.」（ピリオド）が用いられています。逆にいうと、これら以外の文字が入っている場合には、メールアドレスが間違っている可能性があります。これを検出するにはさまざまな方法がありますが、ここでは正規表現を用いて検出を行っています。

スクリプトの動作としては、かなりシンプルにできています。テキストフォームに入力されたメールアドレスに対して、確認する ボタンをクリックしてチェックを行います。



実装としては、確認する ボタンがクリックされたところで、checkAll関数を呼び出しています（①の箇所）。さらに、このcheckAll関数内で、フォームに入力された値を引数にして、checkEmail関数を呼び出しています（②の箇所）。そしてcheckAll関数で、checkEmail関数の戻り値によりエラー処理を行うかどうかを判断します。

checkEmail関数では、まず引数として渡されてきた文字列が格納されている変数strTargetが空でないかどうかチェックします（③の箇所）。空の場合にはfalseを戻り値として返します。空でない場合には、変数strTargetの中に次の文字がないかどうか確認します。

SUPPORT

●確認する特殊文字

!	"	#	$	%	&	'	(	)	=	¥	|	,	:
;	^	[	]	{	}	`	*	?	+	>	<		

確認する方法としては、indexOfメソッドを用いた方法が考えられますが、このスクリプトでは、正規表現を用いた検索を行うことができるmatchメソッドを使っています。matchメソッドは、以下のような書式となります。

```
Stringオブジェクト.match(正規表現)
```

ここでは上記の使ってはいけない文字のどれかにマッチしたら真となるようにするので、次の正規表現を用います。

```
/[検索文字]/
```

このように記述すると、検索文字のうちどれかにマッチすると真となります。実際の記述としては、④の部分のようになります。ここで「¥」が多く使われているのに気付かれた方がいるかと思います。(、)、[、]、{、}、\の7つは、正規表現の中で意味を持っているので、それぞれに「¥」を前に付けて、「¥(」、「¥)」、「¥[」、「¥]」、「¥{」、「¥}」、「¥¥」とすることにより、単なる文字列として見るように指定しています。

ここで、使ってはいけない文字とマッチしたら、checkAll関数に「false」を戻り値として渡すようにしています。検出されない場合は「true」が返されます。checkAll関数では、checkEmail関数の戻り値がfalseの場合には、「メールアドレスが正しくありません」というアラートを表示します。

ところで、このスクリプトではcheckAll関数とcheckEmail関数というように、2つの関数に分けて実装しています。これはWeb上の登録ページにおいて、メールアドレスを入力してもらうフォームが1つしかないわけではなく、複数のメールアドレスを登録してもらう場合があることを考慮しているからです。このスクリプトのようにしておくと、checkEmail関数は汎用的に使えることになります。

なお、このスクリプトでは不正な文字の検出しか行っていません。メールアドレスをきちんとチェックする場合は、さらに「@」やスペースの有無といった検出も組み合わせて使う必要があります。

9-6

WIN e4↑ N4↑ MAC e4↑ N4↑

文字列を確認する3（半角アルファベット）

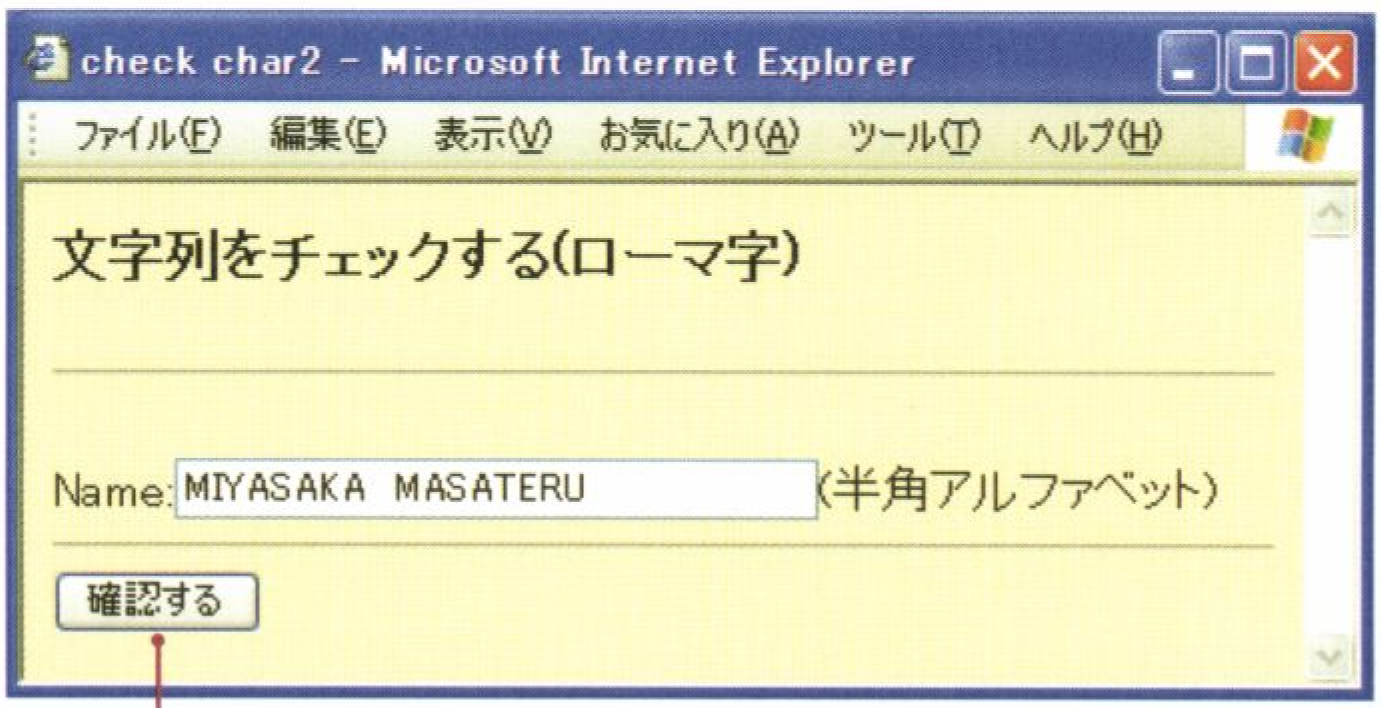
置換後文字列 = Stringオブジェクト.replace(正規表現,"置換文字列")

位置 = 検索対象文字列.indexOf("検索文字",開始位置)

「9-5　文字列を確認する2（特殊記号）」では、メールアドレスに使ってはいけない文字（特殊記号）が入っていないかどうか確認するサンプルを紹介しました。今度は、半角アルファベットで入力しなくてはならない部分に、それ以外の文字が入っていないかどうかを確認するスクリプトです。

半角アルファベットでの名前の入力、希望ログイン名の入力チェックなどに使うことができます。

DISPLAY



全角アルファベットで入力して、
ここをクリックする



SUPPORT

S O U R C E checkchar2.html

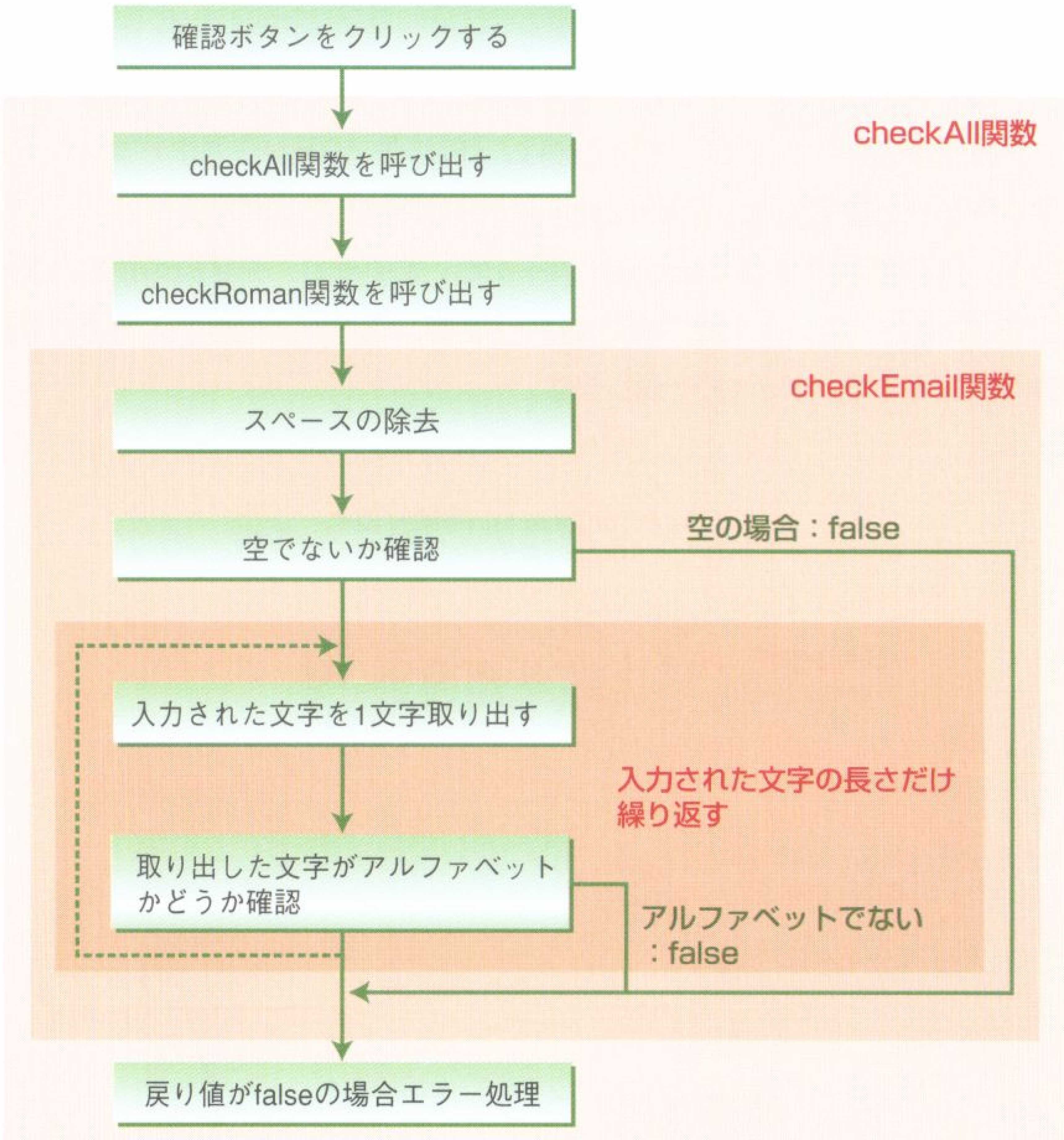
```
<html>
<title>check char2</title>
<script language="javascript">
<!--
function checkAll(){
    if(checkRoman(document.myForm.strRoman.value) == false){ ―①
        document.myForm.strRoman.focus();
        document.myForm.strRoman.select(); ―⑤
        alert("Nameが正しくありません");
        return false;
    }
    else{alert("OK");}
}

function checkRoman(strTarget){
    var i, transTarget;
    var checkStr = new String("ABCDEFGHIJKLMNOPQRSTUVW ┐
⇒      XYZabcdefghijklmnopqrstuvwxyz");                ┘―②
    transTarget = strTarget.replace(/ /g, ""); ―③
    if(transTarget.length==0){ ―④
        return false;
    }
    for(i=0;i<transTarget.length;i++){
        if(checkStr.indexOf(transTarget.charAt(i),0)==-1){
            return false;
        }
    }
    return true;
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>文字列をチェックする(ローマ字)</h3>
<hr />
<form name="myForm">
    Name:<input name="strRoman" type="text" size="40" />(半角アル
⇒      ファベット)<br />
```

```
<hr />
    <input type="button" Value="確認する" onclick="checkAll()" />
</form>
</body>
</html>
```

D E T A I L S

このサンプルでは、入力された項目が半角アルファベットのみであるかどうか確認します。Webページ上にはローマ字で氏名を入力するテキストフォームと、「確認する」と表示されたボタンが配置されています。テキストフォームに入力を行い、 確認する をクリックすると、checkAll関数が呼び出されます。



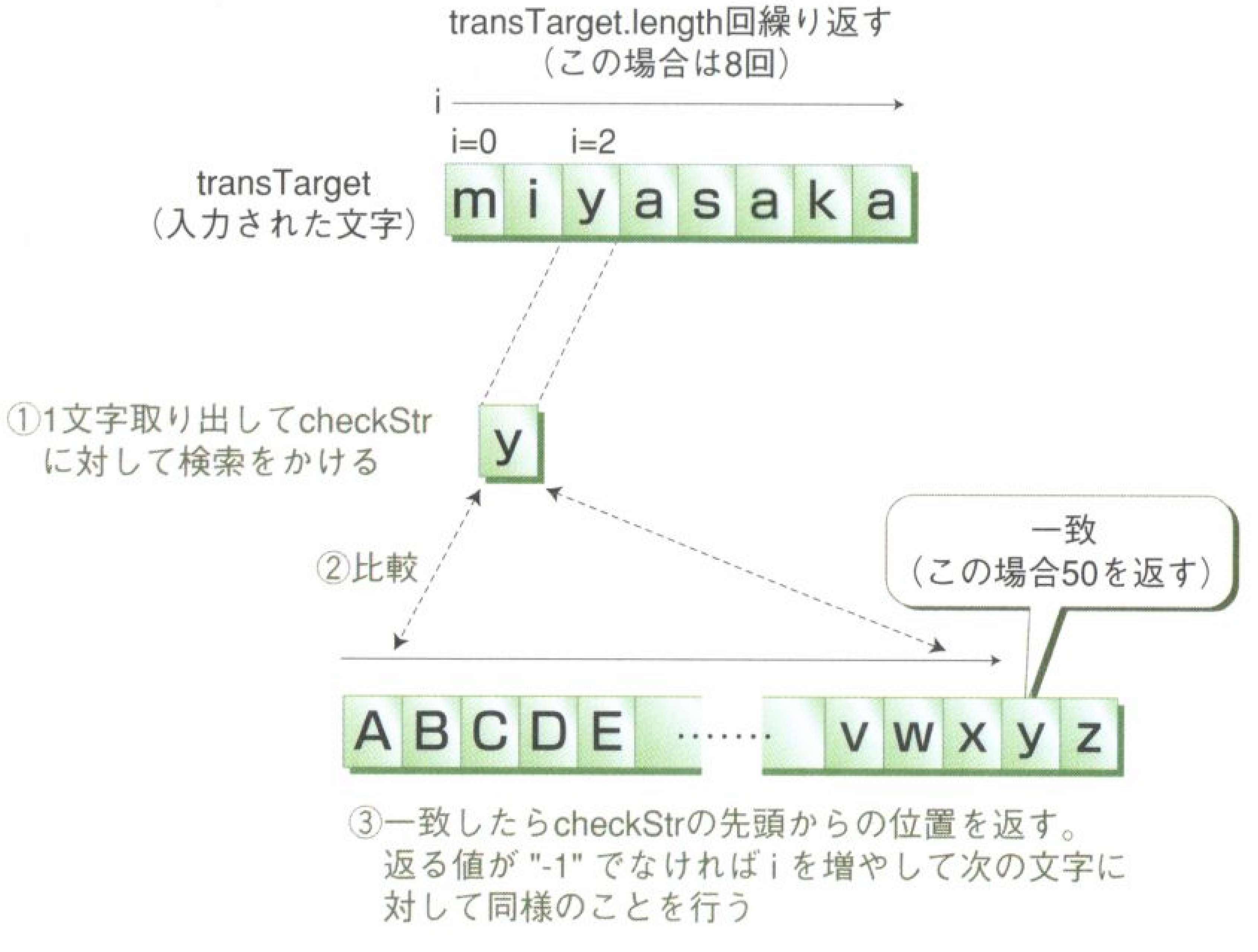
checkAll関数では、①の部分でcheckRoman関数を呼び出しています。if文に囲まれているのは、checkRoman関数の戻り値により処理を変えるためです。checkRoman関数では、②のところで半角アルファベットを定義しています。ここで定義した文字列と入力された文字列を比較し、入力された文字が定義された文字列の中にあるかどうか確認していきます。

ここでのポイントは、確認する前に、入力された文字列からスペースを取り去ることです。これは、スペースだけを入力されたときは、文字列は空であると認識させる必要があるためです。スペースは目には見えませんが、文字として数えられます。そのため、空文字を検出しても、lengthプロパティを使っても、スペースと文字の区別は付かないのです。スペースを取り去るには、replaceメソッドを用います。このメソッドは、正規表現にしたがって文字を置換します。書式は以下のようになります。

```
置換後文字列 = Stringオブジェクト.replace(正規表現,"置換文字列")
```

サンプルでは半角スペースをすべて置換するので、③の部分の記述となっています。置換文字列が「""」になっているのは、「取り去る」という意味です。スペースが取り除かれた文字列を、変数transTargetに格納しています。

この後、④の箇所のようにlengthプロパティを使って、文字列の長さを調べます。「0」の場合には、入力されていなかったか、スペースのみが入力されていたと考えられるので、「false」を返します。



文字の長さを調べた後は、入力されている文字列を調べていきます。入力した文字列を1文字ずつ先頭から切り出し、その文字が②の部分で定義した、変数checkStrの中に存在するかどうかをindexOfメソッドを使って調べます。indexOfメソッドは、以下のような書式になります。

```
検出位置 = 検索対象文字列.indexOf("検索文字", 開始位置)
```

検索対象文字列に検索文字がなかった場合には、「-1」が返されます。そのため、変数transTargetがアルファベット以外だった場合は「-1」が返り、checkAll関数に「false」が返ります。すべての文字を同じように確認して、確認が終了した場合には、checkAll関数にtrueが返ります。

checkRoman関数から戻り値が返ってきたら、値によってcheckAll関数で処理を行います（⑤の箇所）。falseの場合には、フォーカスをローマ字の氏名を入力するフォームに戻し、再度入力しやすくするためにselectメソッドを用いて選択をします。さらにアラートを表示し、再入力を促します。

すでに述べたのですが、checkAll関数とcheckRoman関数が分離されているのは、checkRoman関数を汎用的にし、さまざまなフォーム入力に対して確認のために呼び出すことができるようにするためです。

T I P S

通常このような文字列のチェック処理は、ユーザーに何らかの入力を行ってもらった後に行います。そのため、こういった処理はフォームの「submit」ボタンがクリックされたときに実行されることが多いと思われます。

「submit」ボタンを用いた場合には、P.325の処理と同様に、<form>要素のonclickイベントハンドラの呼び出しに「return」を用いて、呼び出した関数内でエラー処理を行ってください。

「submit」ボタンを用いない場合には、正常に終了したときに、フォームの内容を送信するようにスクリプトを記述します。フォームの内容を送信するにはsubmitメソッドを用います。ここでのスクリプトでは、以下のような記述で送信をすることができます。

```
document.myForm.submit();
```

SUPPORT

9-7 WIN N MAC N

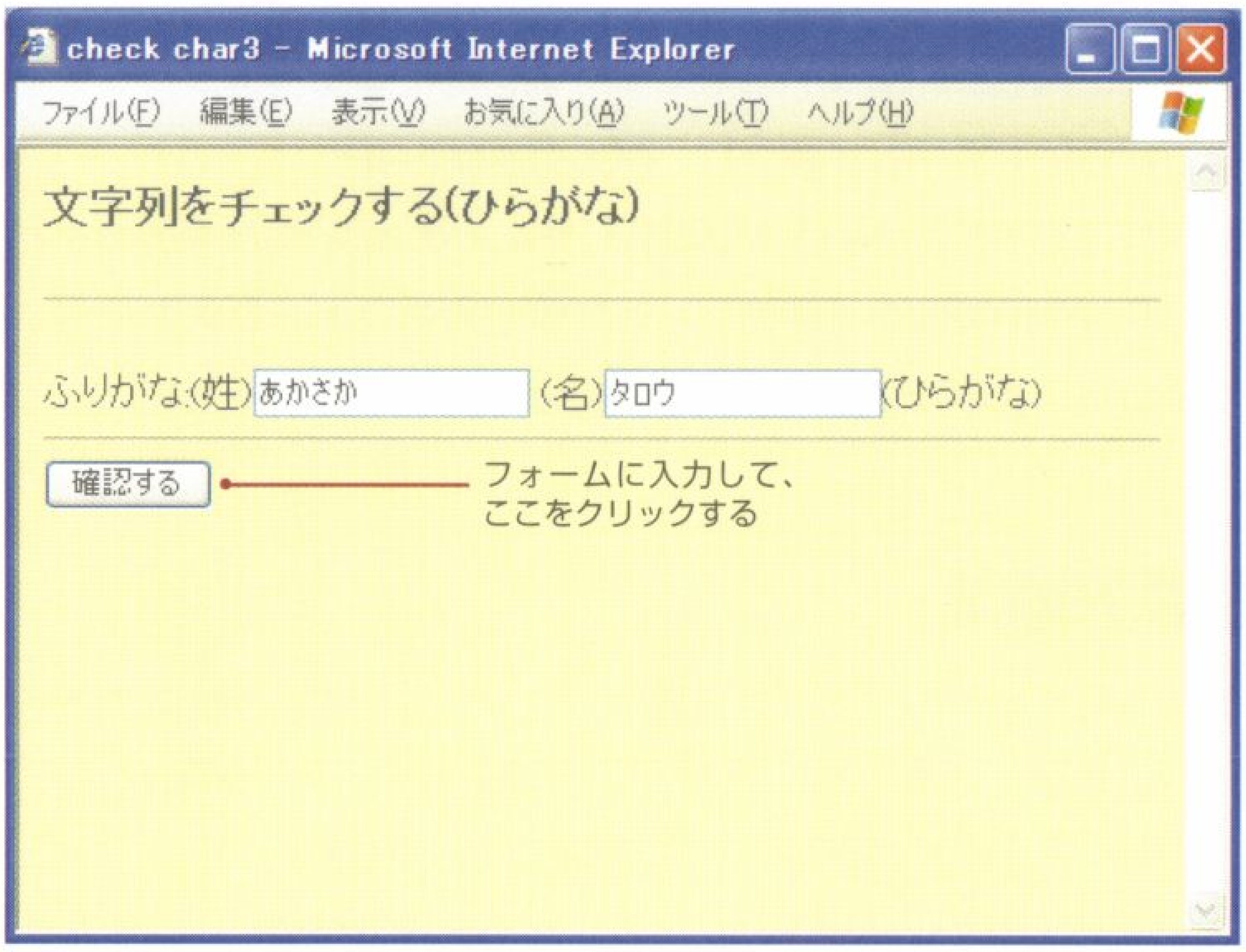
文字列を確認する4（全角ひらがな）

置換後文字列 = Stringオブジェクト.replace(正規表現,"置換文字")

位置 = 検索対象文字列.indexOf("検索文字", 開始位置)

ここまでに、特殊文字の検出と半角アルファベットの確認は解説しました。今度は、日本語の文字列について確認します。Webページで名前を登録させる場合があります。そんなときには、通常、漢字とひらがな、漢字とカタカナで登録させることが多いようです。ここでは、フォームに入力された文字列がひらがなだけかどうかを確認するサンプルを紹介します。

DISPLAY







SUPPORT

SOURCE checkchar3.html

```
<html>
<title>check char3</title>
<script language="javascript">
<!--
function checkAll(){                                   ①
    if(checkHiragana(document.myForm.strHiragana_sei.
        value) == false){
        document.myForm.strHiragana_sei.focus();
        document.myForm.strHiragana_sei.select();
        alert("ひらがな姓が正しくありません");
        return false;
    }
    else{alert("姓OK");}
    if(checkHiragana(document.myForm.strHiragana_na.value) ==
        false){
        document.myForm.strHiragana_na.focus();
        document.myForm.strHiragana_na.select();
        alert("ひらがな名が正しくありません");
        return false;
    }
    else{alert("名OK");}
}

function checkHiragana(strTarget){
    var i, transTarget, char_length;
    var checkStr = new String("ぁあぃいぅうぇえぉおかがきぎ
        くぐけげこごさざしじすずせぜそぞただちぢっつづてでとど
        なにぬねのはばぱひびぴふぶぷへべぺほぼぽまみむめもゃや
        ゅゆょよらりるれろゎわゐゑをん?゛゜ゝゞーー・");          ②
    var char_length = ("あ".length);                    ③

    transTarget = strTarget.replace(/[ 　]/g, "");      ⑤
    if(transTarget.length==0){
        return false;
    }
    for(i=0;i<transTarget.length;i=i+char_length){      ④
        if(checkStr.indexOf(transTarget.charAt(i),0)==-1){
```

▼

SUPPORT

```
            return false;
        }
    }
    return true;
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>文字列をチェックする(ひらがな)</h3>
<hr />
<form name="myForm">
    ふりがな:(姓)<input name="strHiragana_sei" type="text"
⇒       size="20" />
    (名)<input name="strHiragana_na" type="text" size="20" />
⇒       (ひらがな)<br />
<hr />
    <input type="button" value="確認する" onclick="checkAll()" />
</form>
</body>
</html>
```

SUPPORT

D E T A I L S

このサンプルでは、Webページ上に姓、名を入力するテキストフォームと、確認ボタンが配置されています。この 確認する ボタンをクリックすると、テキストフォームに入力された文字列がひらがなかどうかをチェックします。

実はこのサンプルは、「9-6　文字列を確認する3(半角アルファベット)」の半角アルファベットのチェックと基本的な構造は同じです。異なるのは、チェックする入力項目が複数になったため、①の箇所のcheckAll関数内部で、checkHiragana関数を呼ぶ回数が複数回になっている点と、checkHiragana関数の中で、日本語を扱うにあたっていくつか変更を加えている点だけです。

checkAll関数内では、今回は、2回checkHiragana関数を呼び出しています。最初に「姓」のチェックが行われ、次に「名」のチェックが行われます。もしも「姓」のチェックで失敗すると、「名」のチェックは行われずに「姓」のエラー処理が行われてcheckAll関数が終了します。「姓」のチェックが終了すると、「名」のチェックが行われます。

checkHiragana関数では、確認に使う文字列がひらがな(あ～ん～・)になっ

ています。このひらがなをすべて書き出したものが、変数checkStrに格納されます（②の箇所）。さらに、③の部分でひらがな「あ」の文字の長さを確認し、変数char_lengthに格納しています。これはWebブラウザによって全角の長さが異なる可能性があるためです。現在のIE4以降やNN4.06以降ならば、全角は1文字としてカウントされますが、それ以外のWebブラウザでは2文字とカウントされる場合があります。これに伴い、④の部分のfor文の変数iの制御が、単純に「1」を足すだけでなく、変数char_lengthを足していくように変更しています。

もう1箇所、日本語を扱うにあたっての変更が加えられています。紙面ではわかりにくいかもしれませんが、⑤の箇所で行っている、スペースを取り除く処理において、全角スペースも取り除かれるようにしています。

```
transTarget = strTarget.replace(/[ 　]/g, "");
```

正規表現において、[]はその内部に記述されている文字のいずれかとマッチした文字という意味になります。そのため、この[]の中には半角スペースと全角スペースが入力されています。

このサンプルの変数checkStrに設定している文字をカタカナに変更するだけで、カタカナチェックにも用いることができます。

COLUMN

Unicode

コンピュータが扱う情報の中でも、日本語の処理は面倒なものです。コンピュータで文字を扱う場合には、文字コードにする必要があり、日本語を扱う文字コードにはShift-JIS、JIS、EUC、そしてUnicodeがあります。

それぞれの文字コードには一長一短がありますが、インターネットの世界においては、どうやらUnicodeが標準になりそうです。その理由の1つは、XML（eXtensible Markup Language）でUnicodeが推奨されているためです。XMLは電子商取引や電子申請には必須となっています。また、これからはWebサービスのプロトコルとなるであろうSOAPやXML SchemeなどもUnicodeが主に使われるため、さらに普及することでしょう。

また、UnicodeはJavaで採用され、さらにWindows XP/2000/NTの内部コードも、Unicodeで処理されています。

このように、Unicodeはさまざまな分野で広く使われています。Unicodeの情報は、次のURLを参照してください。

URL▶ http://www.unicode.org/

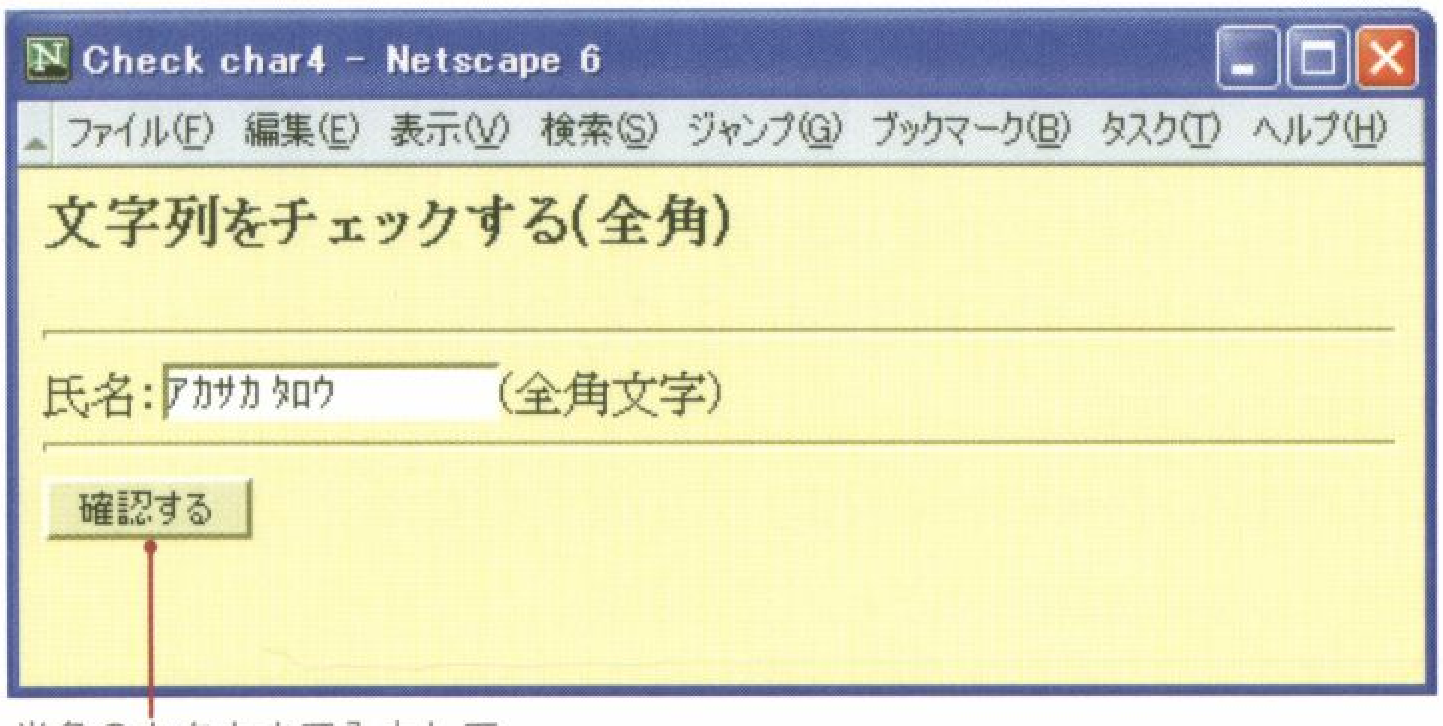
SUPPORT

文字列を確認する5（全角文字）

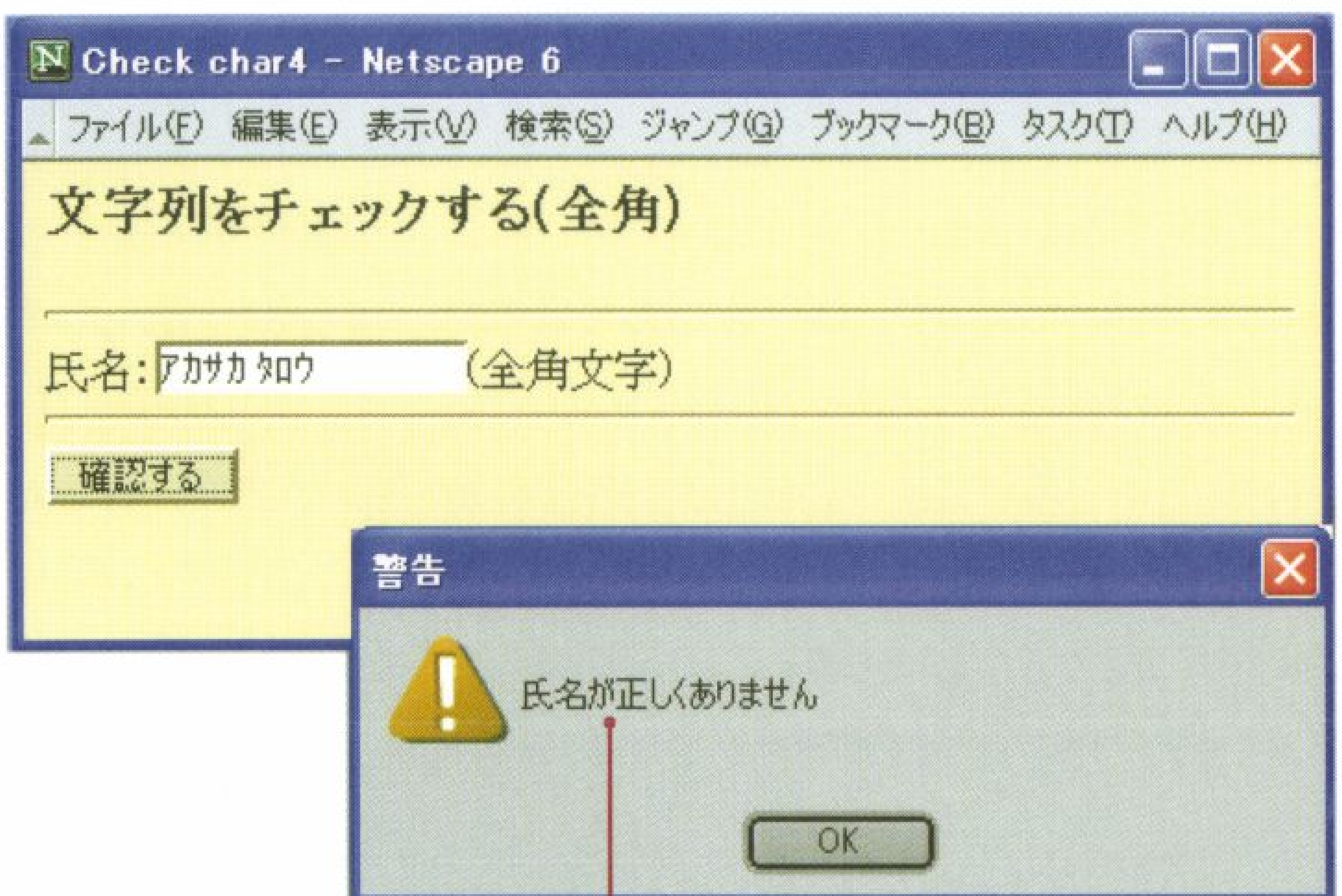
```
if ((比較する文字 >= "!" && 比較する文字 <= "~")||
    (比較する文字 >= "｡" && 比較する文字 <= "ﾟ"))
```

Webページでさまざまな項目を登録する際に、一番多いのは日本語の入力でしょう。皆さんも名前、住所、会社名などを入力したことがあるのではないでしょうか？　フォームによっては、全角で入力を指示している場合があります。そんなときに、全角かどうかをチェックするサンプルを解説します。

DISPLAY



半角のカタカナで入力して、
ここをクリックすると･･･



アラートが
表示される

SOURCE checkzenkaku.html

```
<html>
<title>Check char4</title>
<script language="javascript">
<!--
function checkAll(){
    if(checkZenkaku(document.myForm.strZenkaku.value)
            == false){                                    ②
        document.myForm.strZenkaku.focus();
        document.myForm.strZenkaku.select();
        alert("氏名が正しくありません");
        return false;
    }
    else{ alert("すべて全角です");}
}

function checkZenkaku(strTarget){
    var i, charTarget, transTarget;
    var char_length = ("あ".length);

    transTarget = strTarget.replace(/[   ]/g, "");            ③
    if(transTarget.length==0){
        return false;
    }
    for (i=0; i<transTarget.length; i=i+char_length){
        charTerget = transTarget.charAt(i);
        if ((charTerget >= "!" && charTerget <= "~")||
            (charTerget >= "。" && charTerget <= "°")){        ④
            return false;
        }
    }
    return true;
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>文字列をチェックする(全角)</h3>
<hr />
```

```
<form name="myForm">
    氏名：<input type="text" name="strZenkaku" size="20" />
    (全角文字)<br />
    <hr />
    <input type="button" value="確認する"
⇒      onclick="checkAll()"                                    ①
</form>
</body>
</html>
```

DETAILS

このサンプルでは、Webページに氏名を入力するテキストフォームと確認ボタンが配置されています。 確認する ボタンをクリックすると、テキストフォームに入力された文字列が全角かどうかをチェックします。

Webページの 確認する ボタンがクリックされると、①の部分のonclickイベントハンドラで指定されているcheckAll関数が呼び出されます。そして、checkAll関数の中のif文の条件式で、さらにcheckZenkaku関数を呼び出しています。これは、checkZenkaku関数の戻り値によりエラー処理を行うためです。

checkZenkaku関数では、変数の宣言を行った後、「あ」の文字列の長さを確認しています。これはP.293でも解説しているように、Webブラウザによって全角文字の長さのカウント方法が異なる可能性があるからです。

次に、全角スペースと半角スペースを除去し、入力された文字列の長さの確認をしています(③の箇所)。ここで長さが「0」の場合には、checkAll関数に「false」を返して終了します。

この後、スペースが除去された文字列を格納した変数transTargetに対して、1文字ずつ切り出して、全角文字かどうかを確認します。確認には逆に、半角文字でないかどうかを調べています。④の部分で記述されているのは、「!」から「~」の間、または「。」から「゜」の間の場合には、切り出された文字を半角文字と見なして、checkAll関数に「false」を返すようになっています。これはどういうことかというと、文字コード上で考えると「!」から「~」の間と「。」から「゜」の間は、半角の文字コードとなっているからです。そこで、大小の条件式でこの範囲に文字かどうかを確認しているのです。

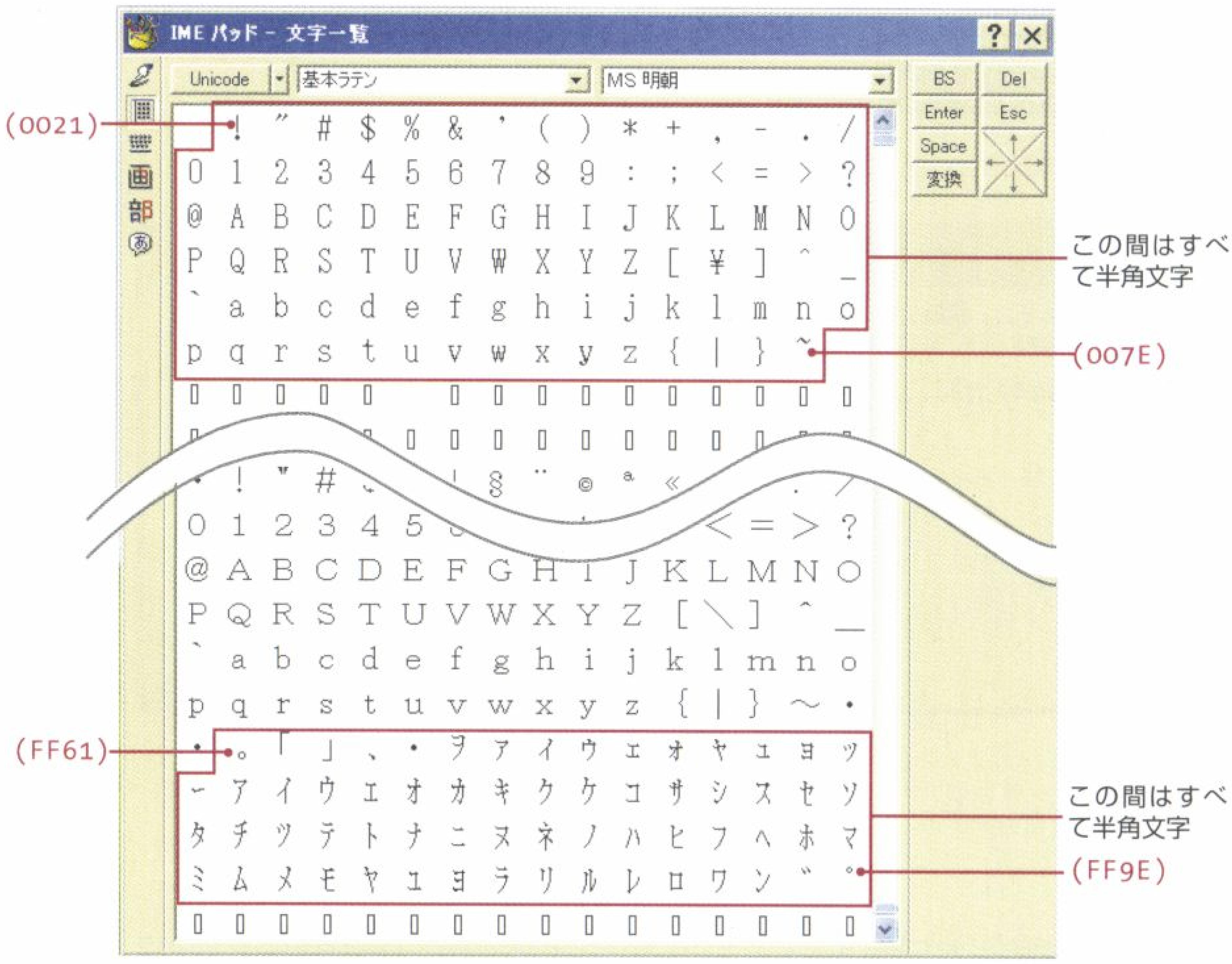
それぞれの文字の、Unicodeでの文字コードは、次のようになります。

SUPPORT

文字	文字名	16進	10進
!	Exclamation Mark	0021	33
~	Tilde	007E	126
｡	Halfwidth Ideographic Full Space	FF61	65377
ﾟ	Halfwidth katakana Semi-voiced Sound Mark	FF9F	65439

このように文字コード上の文字の範囲を使うことにより、文字列の確認をすることができます。

▼文字コード表での範囲



checkAll関数では、checkZenkaku関数からの戻り値が「false」の場合には、対象となったテキストフォーム（このスクリプトでは"document.myForm.strZenkaku"）にフォーカスを合わせ、入力文字列をハイライトし、アラートでエラーメッセージを出力します。

このように、たった1つの文字列の入力にしても、さまざまな点で気を付けなくてはならない部分があります。しかし、このような処理をしていないと、インターネットショップやアンケートなどでトラブルの原因となりますから、よく考えて利用するようにしてください。

9-9 WIN IE 4↑ N 4↑ MAC IE 4↑ N 4↑

入力サポート

文字列 + document.myForm.myText.value

document.myForm.myText.value + 文字列

最近では電子商取引(EC)も盛んになってきました。その際によく自分の住所、勤務先を入力するといった場面に遭遇します。そんなときに、入力の補助をしてくれたら、と思うことはありませんか？　そんな機能を実現するサンプルになります。

DISPLAY

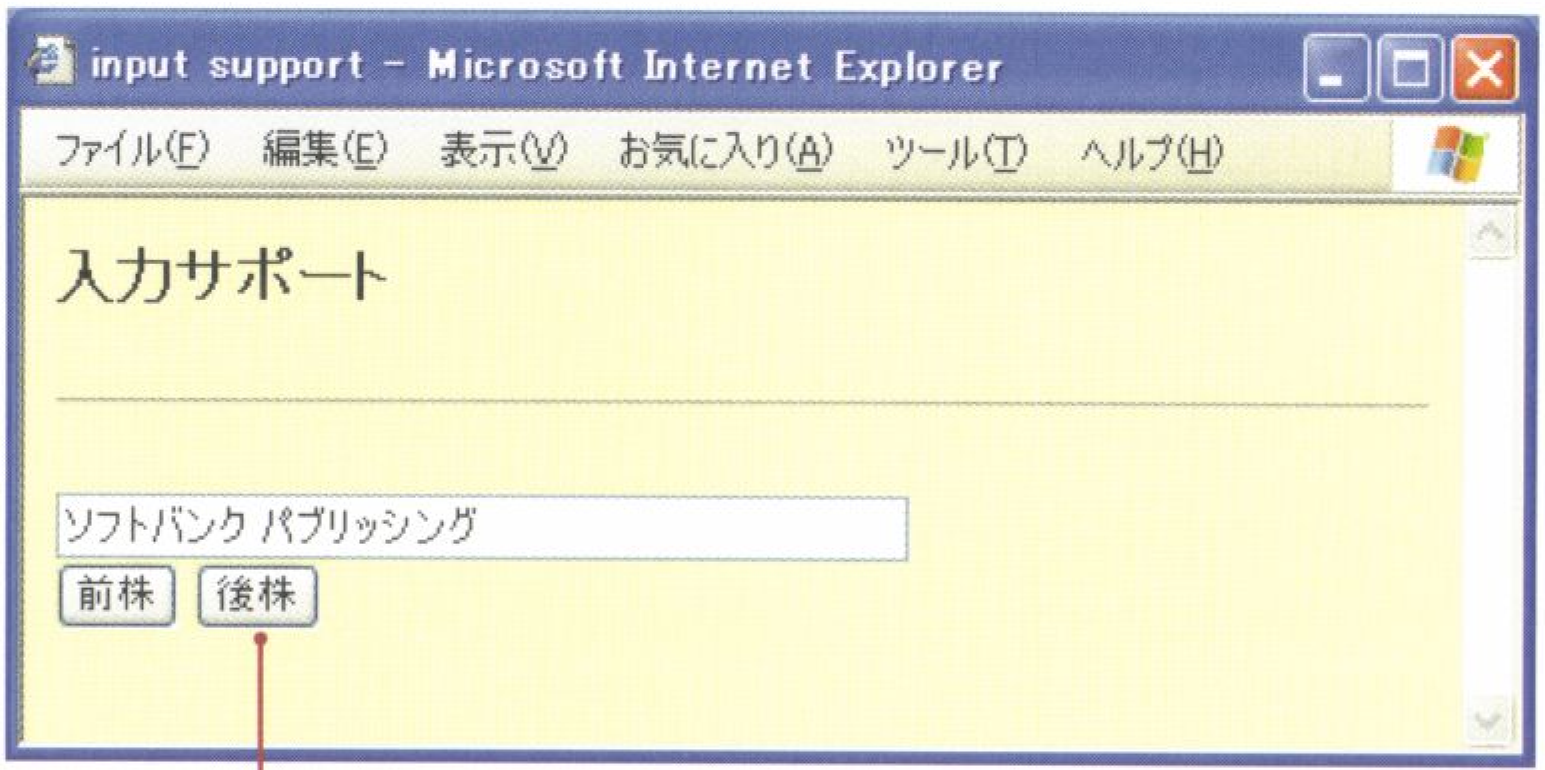
▼前に文字列を付加する



ここをクリックして、
前に文字列を付加する



▼後に文字列を付加する



ここをクリックして、
後に文字列を付加する



SOURCE

inputsapport..html

```
<html>
<title>input support</title>
<script language="javascript">
<!--
function addText(position, addtext){  ①
if(position == "before"){
      res = addtext + document.myForm.myText.value;
   }
   else{
      res = document.myForm.myText.value + addtext;
   }
   document.myForm.myText.value = res;
```

▼

```
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>入力サポート</h3>
<hr />
<form name="myForm">
    <input type="text" name="myText" size="50" /><br />
    <input type="button" value="前株" onclick="addText('before',
⇒       '株式会社')" />
    <input type="button" value="後株" onclick="addText('after',
⇒       '株式会社')" />
</form>
</body>
</html>
```

DETAILS

このサンプルでは「株式会社」という言葉をテキストフォームの入力項目に対して付加することができます。Webページ上には2つのフォームボタンが用意されており、それぞれ「前株」、「後株」というように表示されています。それぞれのボタンがクリックされるとフォーム内の文字列に対して「株式会社」という文字列が前、または後ろに付加されます。

この文字列の付加を実現している部分は、①で記述しているaddText関数です。addText関数は、2つ引数を取ります。最初の引数positionは付加する文字列の位置、2つ目の引数addtextには付加する文字列を取ります。addText関数内では、引数として取ったpositionの値を見て、「before」ならばフォーム内の文字列の前に、「after」なら文字列の後ろに変数addtextに格納されている文字列を連結します。文字列の連結には「+」演算子を用います。そして、連結した文字列をフォームに対して代入しています。

このスクリプトでは、addText関数に対して2つの引数を渡しているため、汎用的に使える関数になっています。連結する文字列を変更する場合には、関数の呼び出し側の2つ目の引数を書き換えればよいことになります。

9-10 計算をリアルタイムで行う

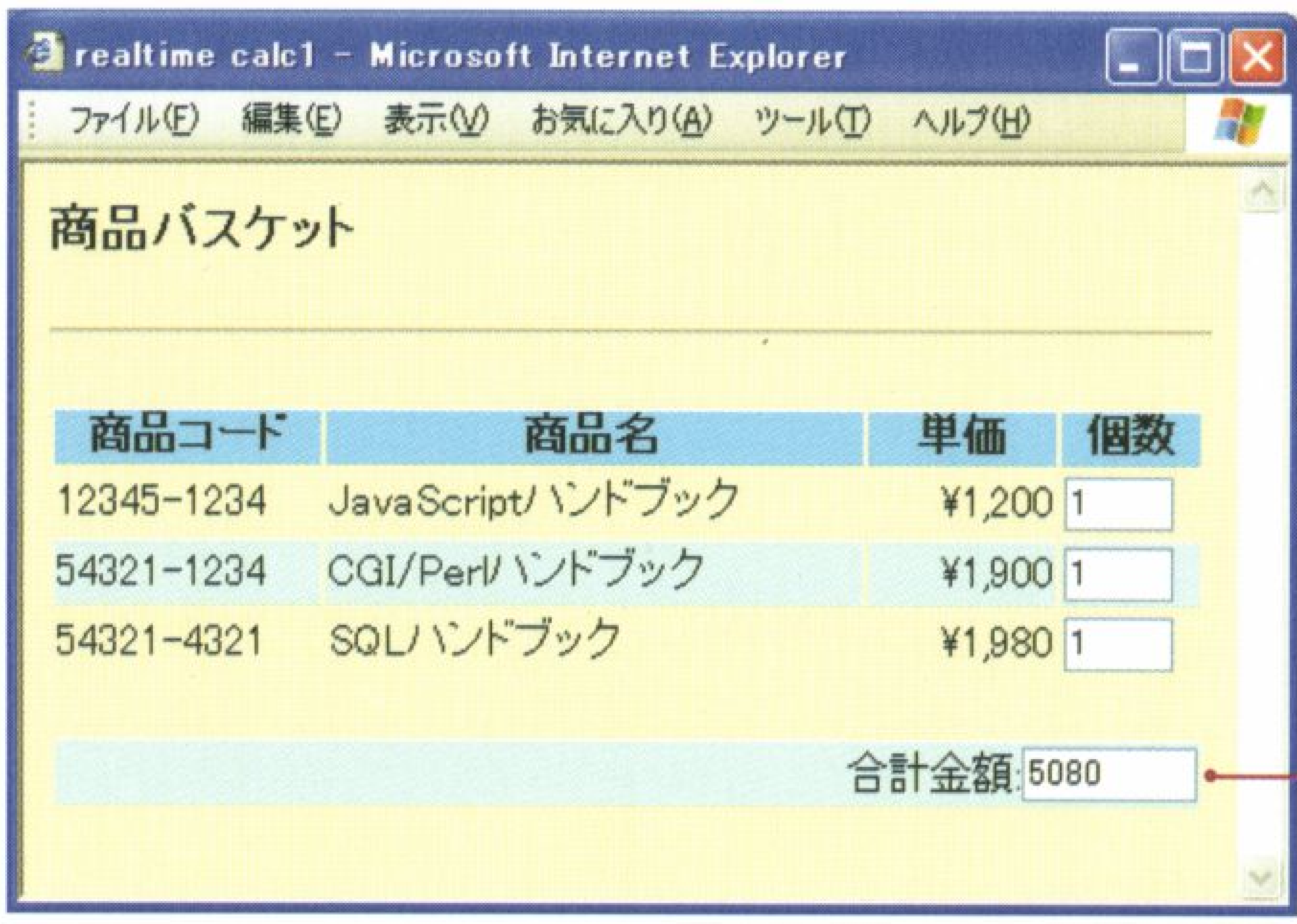
WIN e4↑ N4↑ MAC e4↑ N4↑

`document.formオブジェクト.elements[++i].value`

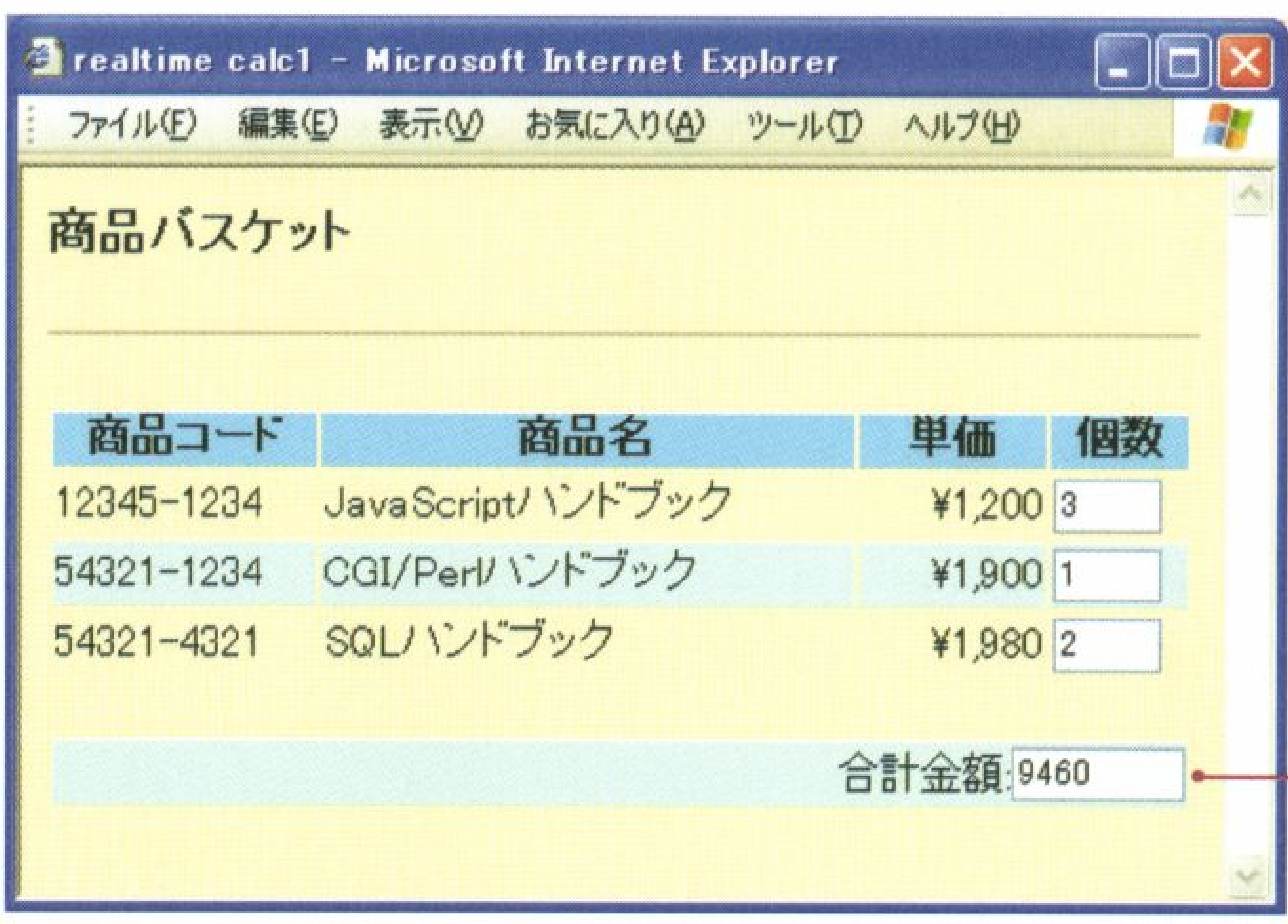
`<Input type="hidden" …>`

インターネット上のショップで買い物をしているときに、ショッピングカートや見積もりの画面を見たことがあると思います。そういったWebページでは、品物の個数を変えたり、オプションを変更したりするとすぐに合計金額が変更されるものが数多くあります。そんなWebページを実現するための基礎的な部分を解説してみます。

D I S P L A Y



読み込み時に、各1冊の合計が計算される



冊数を変更すると、リアルタイムに合計が変更される

SOURCE realtimecalc.html

```
<html>
<title>realtime calc1</title>
<script language="javascript">
<!--
var amount = 3;

function calcTotal(){
   var total=0;
   var i=0;
   while(i<amount*2){ ―――――――――――――――――――――――――――― ①
      total = total + document.myForm.elements[i]
➡        .value*document.myForm.elements[++i].value;
      i++; ――――――――――――――――――――――――――――――――――――― ②
   }
   document.myForm.itotal.value = total; ―――――――――――― ③

}
//-->
</script>
<body onload="calcTotal()" bgcolor="#ffffcc">
<h3>商品バスケット</h3>
<hr />
<form name="myForm" method="post">
<table>
<tr bgcolor="#66ccff">
   <th width="100">商品コード</th>
   <th width="200">商品名</th>
   <th width="70">単価</th>
   <th width="50">個数</th>
</tr>
<tr>
   <td>12345-1234</td>
   <td>JavaScriptハンドブック</td>
   <td align="right">¥1,200<Input type="hidden" name="row0v"
➡     value="1200"></td>
   <td><Input type="text" name="row0" value="1" size="5"
➡     onChange="calcTotal()"></td>
```

▼

SUPPORT

```
</tr>
<tr bgcolor="ccffff">
   <td>54321-1234</td>
   <td>CGI/Perlハンドブック</td>
   <td align="right">¥1,900<Input type="hidden" name="row1v"
⇒     value="1900"></td>
   <td><Input type="text" name="row1" value="1" size="5"
⇒     onChange="calcTotal()"></td>
</tr>
<tr>
   <td>54321-4321</td>
   <td>SQLハンドブック</td>
   <td align="right">¥1,980<Input type="hidden" name="row2v"
⇒     value="1980"></td>
   <td><Input type="text" name="row2" value="1" size="5"
⇒     onChange="calcTotal()"></td>
</tr>
<tr><td colspan="4"> </td></tr>
<tr bgcolor="#ccffff">
   <td colspan="4" align="right">合計金額:<Input type="text"
⇒     name="itotal" value="0" size="10"></td>
</tr>

</table>
</form>
</body>
</html>
```

DETAILS

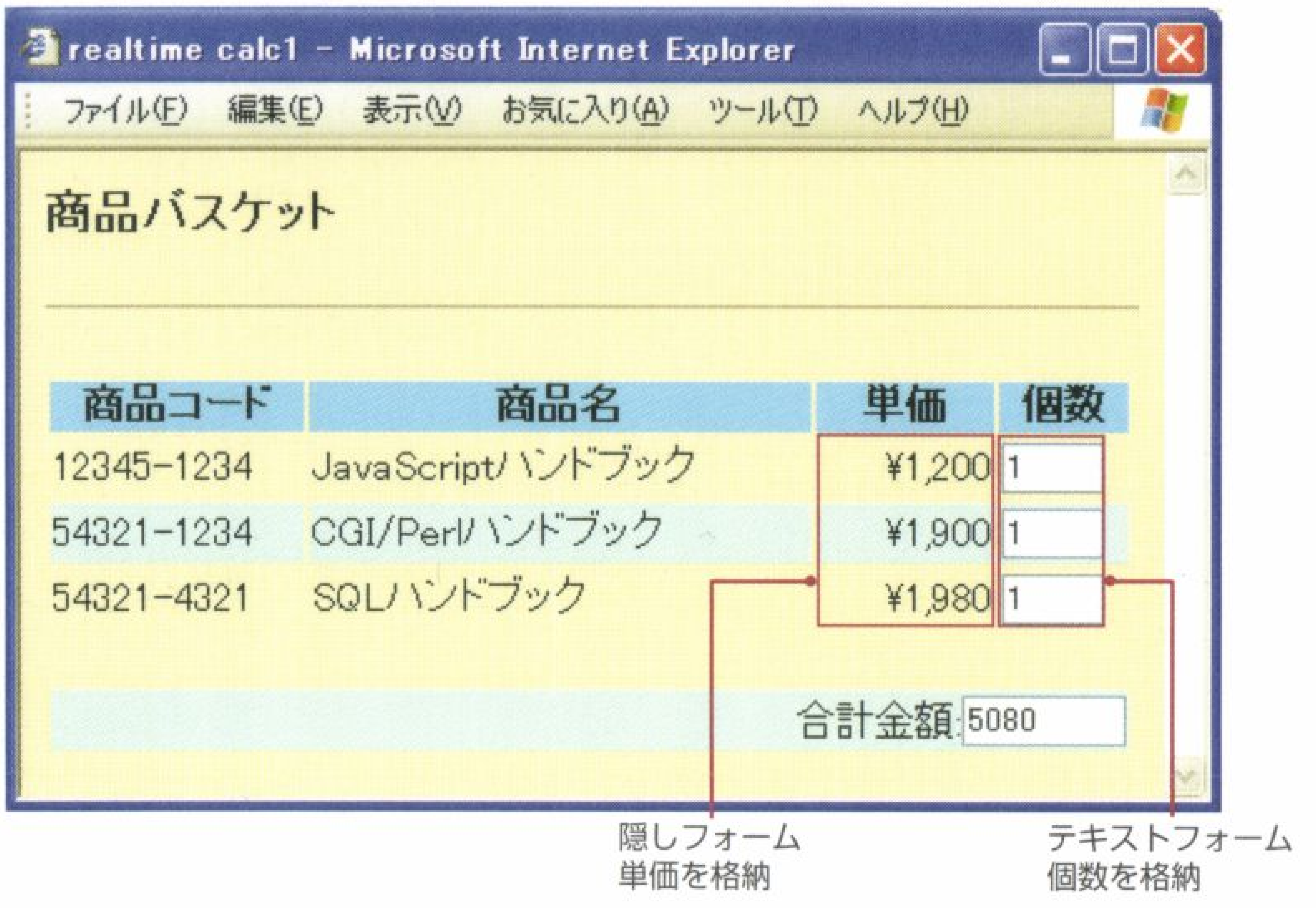
サンプルでは、Webページのロード時と、テーブル内の品物の個数を変更すると、合計金額が即座に計算され、表示されるようになっています。計算はcalcTotal関数に実装されています。

calcTotal関数の呼び出しは、Webページロード時にonloadイベントハンドラを用いて呼び出され、テキストフォーム内の数値が変更されたときにonchangeイベントハンドラを用いて呼び出されています。関数の呼び出しには引数はありません。

SUPPORT

calcTotal関数では、テキストフォーム内の品物の個数と単価を使って合計金額を計算します。計算は各行の単価と個数を掛け合わせて行の小計を算出し、行の小計をすべて足し合わせたものを合計としています。こういった計算をする際のヒントとなる部分が、この中に隠されています。それは、Webページには表示されていないフォームが埋め込まれているということです。

商品の個数は、Webページにテキストフォームとして表示されていますが、単価はフォームとしては表示されていません。しかし、フォームでない場合は、通常、Webページに表示されていたとしても計算に使うことはできません。そこで、ソースコードを見るとわかるようにhiddenフォームが埋め込まれていて、それに「rowNv」(Nは行から1を引いた数字)という名前が付けられています。値としては、その行に表示されている商品の単価が格納されています。



ここで、隠しフォームまで含めてフォームの数を数えると、行(変数amountに格納)の2倍のフォームがあることになります。そこで、①の記述のようにwhileの条件式は行の2倍までとなります。

実際の計算では、elements[i]とelements[i+1]を掛けると行ごとの小計が算出できることになります。なお、このサンプルでは、elements[i+1]を作るためにelements[++1]としています。これは、変数iに「1」を足してから計算に使うという意味です。さらに、whileループの先頭に戻る前に変数iに1を足しています(②の箇所)。これでwhileループを1回処理するたびに、変数iは「2」ずつ増えていくことになり、whileループは行の回数だけ繰り返されます。なお、

whileループ中で各行の小計が変数totalに足し合わされているので、whileループが終了するときには合計値が算出されていることになります。

最後に、算出した合計値をitotalと名付けられたフォームに表示して終わります(③の箇所)。また、このサンプルでは実装していませんが、消費税の算出や送料を足す場合には、この後に処理を追加することになります。

実際にWebページに実装しようと考えた場合、ここで紹介したサンプルはそのままで使えるというものではありません。というのは、こういったWebページを静的に用意しても意味がないからです。

T I P S

実際には、ユーザーからの反応によって、CGIやASP(Active Server Pages)を使って、Webサーバー側で処理を行い、その結果としてこのスクリプトのようなWebページが生成されてくるのが通常のシステムになります。そのため、CGIやASPの記述の中に、ここで解説したスクリプトの要素を埋め込んでいくことになります。

ちょっと考えると、サーバーサイドで動作するスクリプト(CGIやASPなど)とクライアントサイドで動作するスクリプト(本サンプルのようなスクリプト)が混在することになり、なかなか厄介に感じるでしょう。しかし、ユーザーにとってよりよいWebページを作るために、何とか乗り切ってください。

9-11 WIN N MAC N

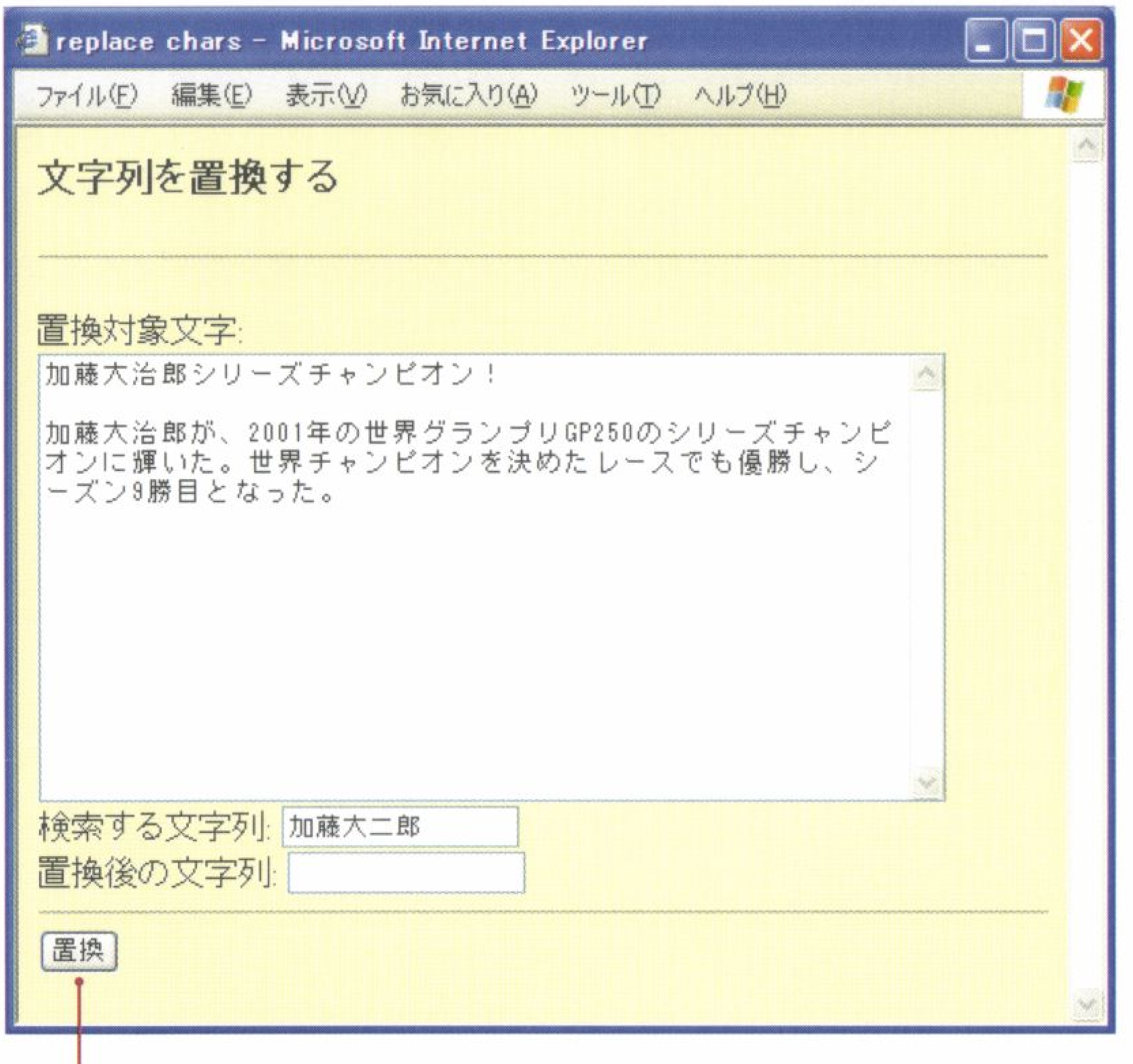
文字列を置換する

```
変数 = 置換対象テキスト.replace(RegExpオブジェクト, 置換文字列);
```

文字列を扱う作業をしているときに、よく文字列置換を行います。この非常に機能を限定したものを、Webページ上で作成してみます。

DISPLAY

▼テキストボックスの1つでも入力しなかった場合

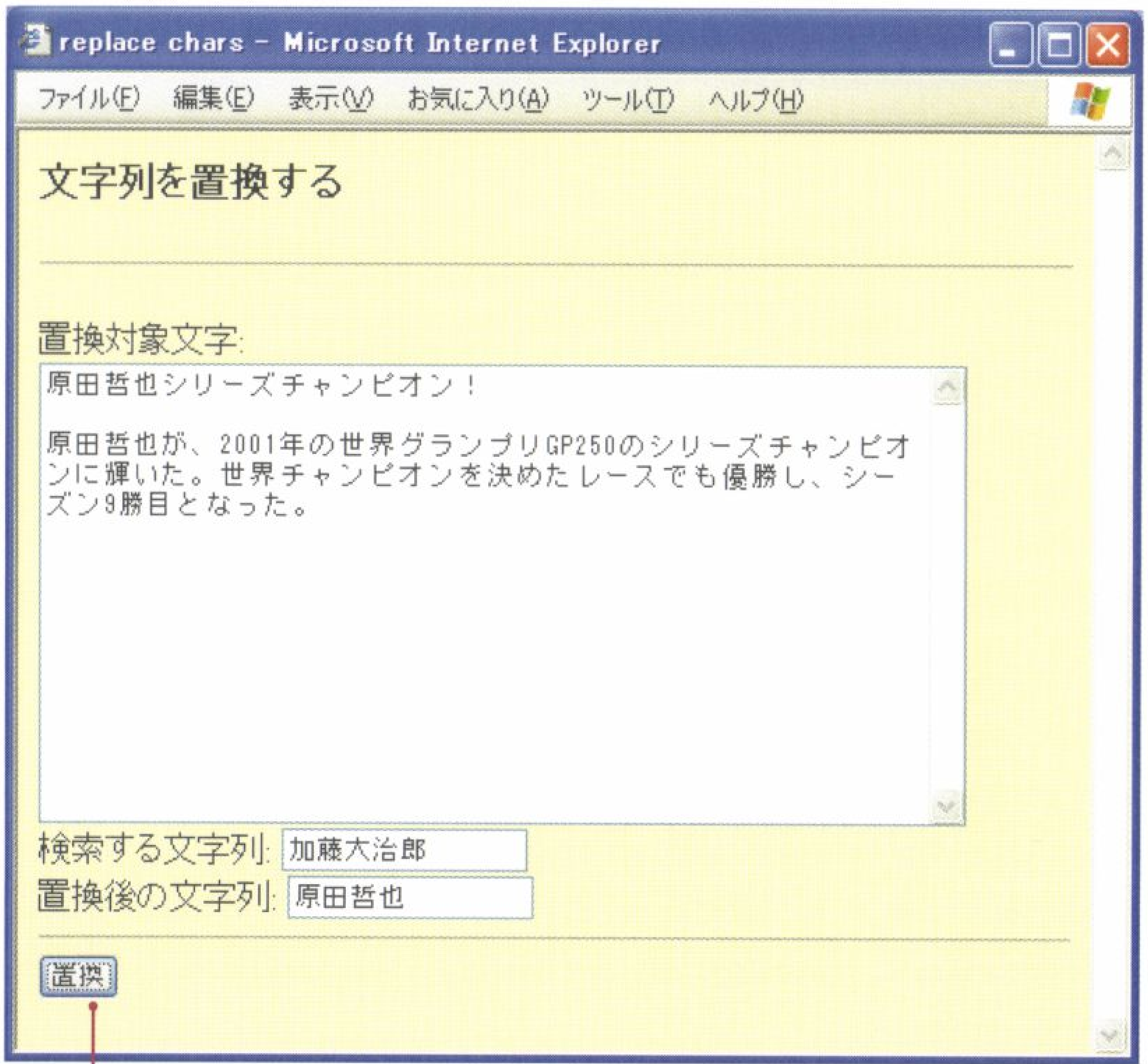


テキストボックスに1つでも入力せずに、ここをクリックすると･･･



アラートが表示される

▼すべてテキストボックスに入力した場合



すべて入力してここをクリックすると、検索する文字列が置換後の文字列に置換される

SOURCE replacechar.html

```
<html>
<title>replace chars</title>
<script language="javascript">
<!--
function replaceChars(){
   var strTarget, resultStr;

   //check form
   if(document.myForm.myText.value == ""){
      alert("検索対象がありません");
      document.myForm.myText.focus();
      return;
   }
```

```
    if(document.myForm.sText.value==""){
        alert("検索する文字列がありません");
        document.myForm.sText.focus();
        return;
    }
    if(document.myForm.rText.value==""){
        alert("置換後の文字列がありません");
        document.myForm.rText.focus();
        return;
    }

    //replace process
    var regPattern = new RegExp(
➡       document.myForm.sText.value, "g"); ――――――――②
    var replaceStr = document.myForm.rText.value;
    var targetStr = document.myForm.myText.value;
    var resultSrt = targetStr.replace(                     ┐
➡       regPattern, replaceStr);document.myForm.myText.value ├③
➡       = resultSrt;                                       ┘
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>文字列を置換する</h3>
<hr/>
<form name="myForm">
    置換対象文字:<br />
    <textarea name="myText" cols="60" rows="15"></textarea>
➡       <br />
    検索する文字列: <input name="sText" type="text" size="20" />
➡       <br />
    置換後の文字列: <input name="rText" type="text" size="20" />
➡       <br />
    <hr />
    <input type="button" value="置換"
➡       onclick="replaceChars()" />
</form> ――――――――――――――――――――――――――――――――――――――①
</body>
</html>
```

SUPPORT

DETAILS

このスクリプトでは、大きなテキストエリアと、テキストボックスが2つ、さらに 置換 ボタンが用意されています。

動作としては、「置換対象文字列」テキストエリアに文字列を入力します。さらに「検索する文字列」に検索した文字、「置換後の文字列」に検索した文字列を置き換える文字列を入力します。そして、 置換 ボタンをクリックすると置換対象文字列のテキストで、「検索する文字列」に入力した文字列と同じ部分が「置換後の文字列」にすべて置換されます。

動作としては 置換 ボタンをクリックすると、onclickイベントハンドラに指定されたreplaceChars関数が起動されます(①の箇所)。replaceChars関数は大きく分けて2つの部分で構成されています。1つはWebページ上に配置されたフォームの入力チェック、もう1つは実際に文字列を置換している部分です。

フォームのチェック部分では、それぞれ3つのフォームが空でないかどうかを確認しています。もしも、フォームに入力がない場合には、それぞれにアラートを表示し、そのフォームにフォーカスを移動します。さらに、returnを返すのでreplaceChars関数をその部分で終了します。そのため、文字列の置換は行われません。

特定のフォームにフォーカスを移動するにはfocusメソッドを用います。focusメソッドの書式は以下のようになります。

```
documentオブジェクト.formオブジェクト.要素オブジェクト.focus();
```

このように記述すると、指定された要素オブジェクトがフォーカスされます。

次に、文字列の置換する部分です。文字列の置換にはRegExpオブジェクトとreplaceメソッドが用いられています。RegExpオブジェクトは検索のパターンを指定するものです。書式は以下のようになります。

```
var 変数 = new RegExp(検索パターン[, "フラグ"]);
```

●RegExpオブジェクトのフラグ

g	:	すべてのパターンを変換する
i	:	大文字と小文字を区別しない

このフラグの指定は省略することができます。また、上記の記述は、次のように記述することもできます。

SUPPORT

```
変数 = /検索パターン/フラグ;
```

スクリプトでは、「検索する文字列」と名付けられているテキストフォームから取得した値を検索パターンに入れ、フラグで「g」を指定しています（②の箇所）。そのため、すべてのパターンを変換する指定になっています。

検索パターンを設定した後は、実際に置換をする部分です。置換にはreplaceメソッドを用います。replaceメソッドは以下のような書式となります。

```
変数 = 置換対象テキスト.replace(RegExpオブジェクト,
置換文字列);
```

そのため、置換が行われるテキストである「置換対象文字」テキストエリアの値が変数targetStrに、置換文字列が変数replaceStrに格納され、③の箇所のように、replaceメソッドが実行されます。replaceメソッドの実行結果は左辺の変数resultSrtに格納されます。最後に置換結果の変数をテキストエリアに代入しています。

このように比較的簡単に、テキストを置換するWebページを制作することができます。検索条件の指定のためのラジオボックスなどを付けるとより一層使い勝手のよいものが作成できるでしょう。

10-1 WIN 4↑ N4 MAC 4↑ N4

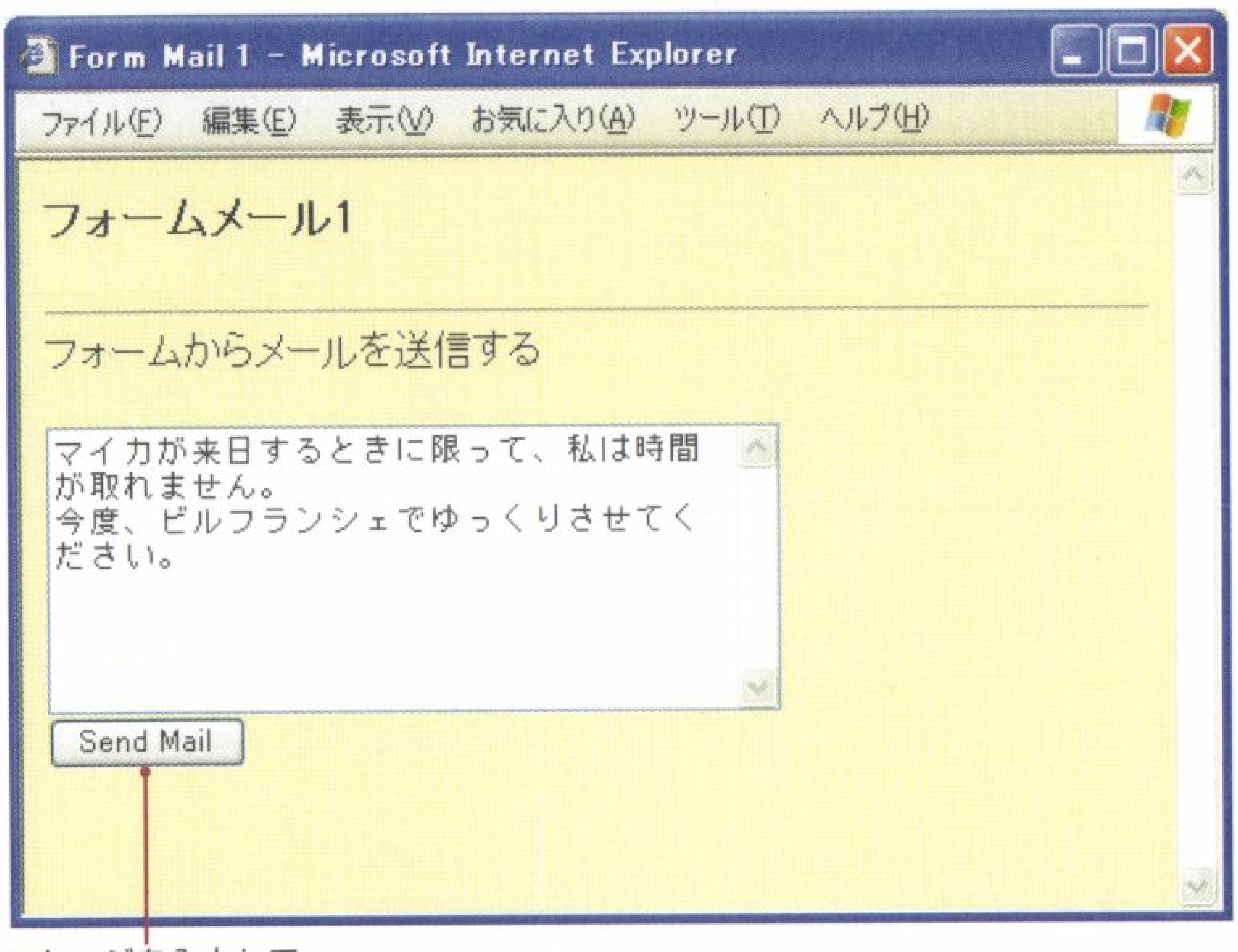
フォームからメールを送信する

```
<form name= "form1"  method= "post"  action=
"mailto:送信先メールアドレス"  onsubmit= "document.
formオブジェクト名.encoding= 'text/plain'" >
```

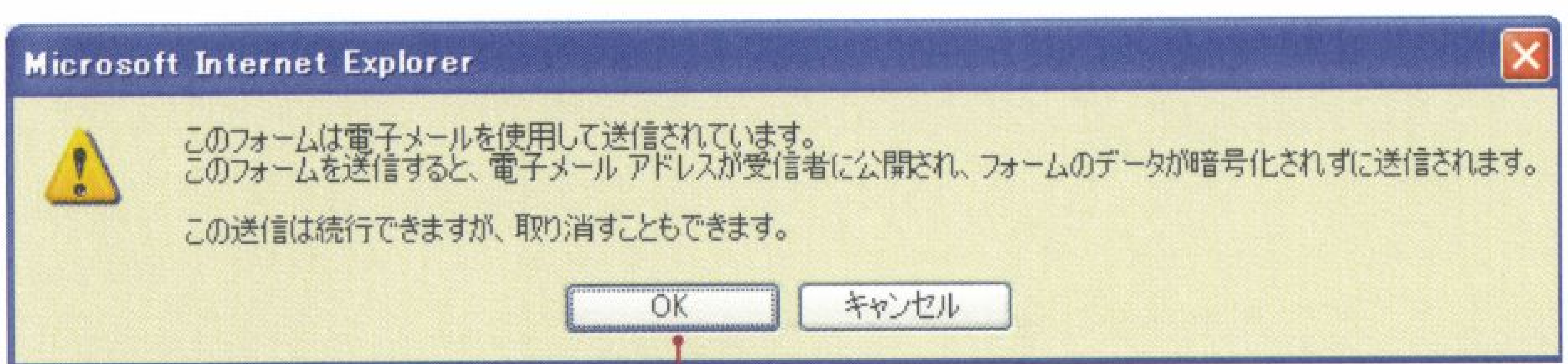
Webページで、ユーザーから感想などのフィードバックを送ってもらう方法の1つにメールがあります。しかし、Webページを見ている途中で、メールを書くためにわざわざメールソフトを起動するのは、ユーザーにとって面倒なことです。そんなときに、JavaScriptを使って、Webページ上から直接メールを送信するようにしておくと便利でしょう。こうしておけば余計な手間もかからず、ユーザーに親切なWebページになります。

DISPLAY

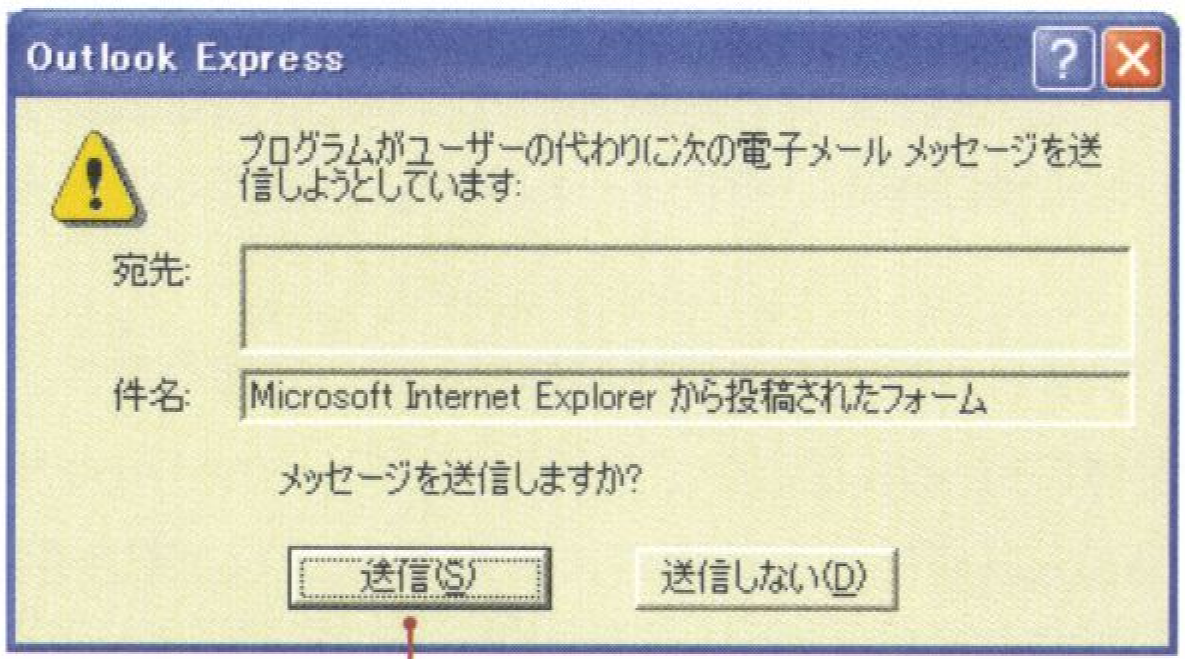
▼フォームメールの送信



メッセージを入力して、
ここをクリックする



クリックして、操作を続行する



ここをクリックして送信する

▼受信したメールソフトでの表示



メールソフトで受信したメッセージ

SOURCE　　formmail1.html

```
<html>
<title>Form Mail 1</title>
<body bgcolor="#fff8dc">
<h3>フォームメール1</h3>
<hr />
フォームからメールを送信する<br />
<form name="form1" method="post"
⇒  action="mailto:address@domain"
⇒  onsubmit="document.form1.encoding='text/plain'">
    <textarea rows="8" cols="40" name="mail"></textarea><br />
    <input type="Submit" value="Send Mail">
</form>
</body>
</html>
```

DETAILS

このサンプルは、テキストエリアに文章を入力し、[Send Mail]ボタンをクリックすると、入力した文章が指定されたメールアドレスに対して送られるというものです。

サンプルでは、フォームにさまざまな属性が指定してあります。まず、name属性でフォーム全体に「form1」という名前を設定し、method属性でpost(標準入力)の指定、action属性で送信先のメールアドレスの指定を行っています。メールアドレスの指定は以下のようになります。

```
action="mailto:送信先メールアドレス"
```

また、[Send Mail]ボタンをクリックすると、<form>タグのonsubmitイベントハンドラによってフォームの送信形式が設定されます。これは以下のように指定します。

```
document.formオブジェクト名.encoding
```

このサンプルでは、「text/plain」に設定しています。

ここまでの処理により、テキストエリアに入力された文章が、<form>タグのaction属性に設定してあるメールアドレスに、メールとして送信されます。なお、通常は標準のメールソフトの送信トレイに入れられます。

TIPS

このサンプルのように、JavaScriptを用いてWebページからメールを送信する場合には、使っているWebブラウザとメールソフトの関係によってうまく動作しないことがあります。

IEの場合、標準で使用するメールソフトがOutlook Expressでないと、使用しているメールソフトのメール作成画面が起動してしまいます。NN4の場合、標準で使用するソフトがNetscape Mail(Messenger)でないと使用しているメールソフトのメール作成画面が起動してしまいます。

そして、NN6ではセキュリティの関係から、標準で使用するメールソフトがNetscape Mailであっても、メールソフトのメール作成画面が表示されます。

10-2

WIN 4↑ N4 MAC 4↑ N4

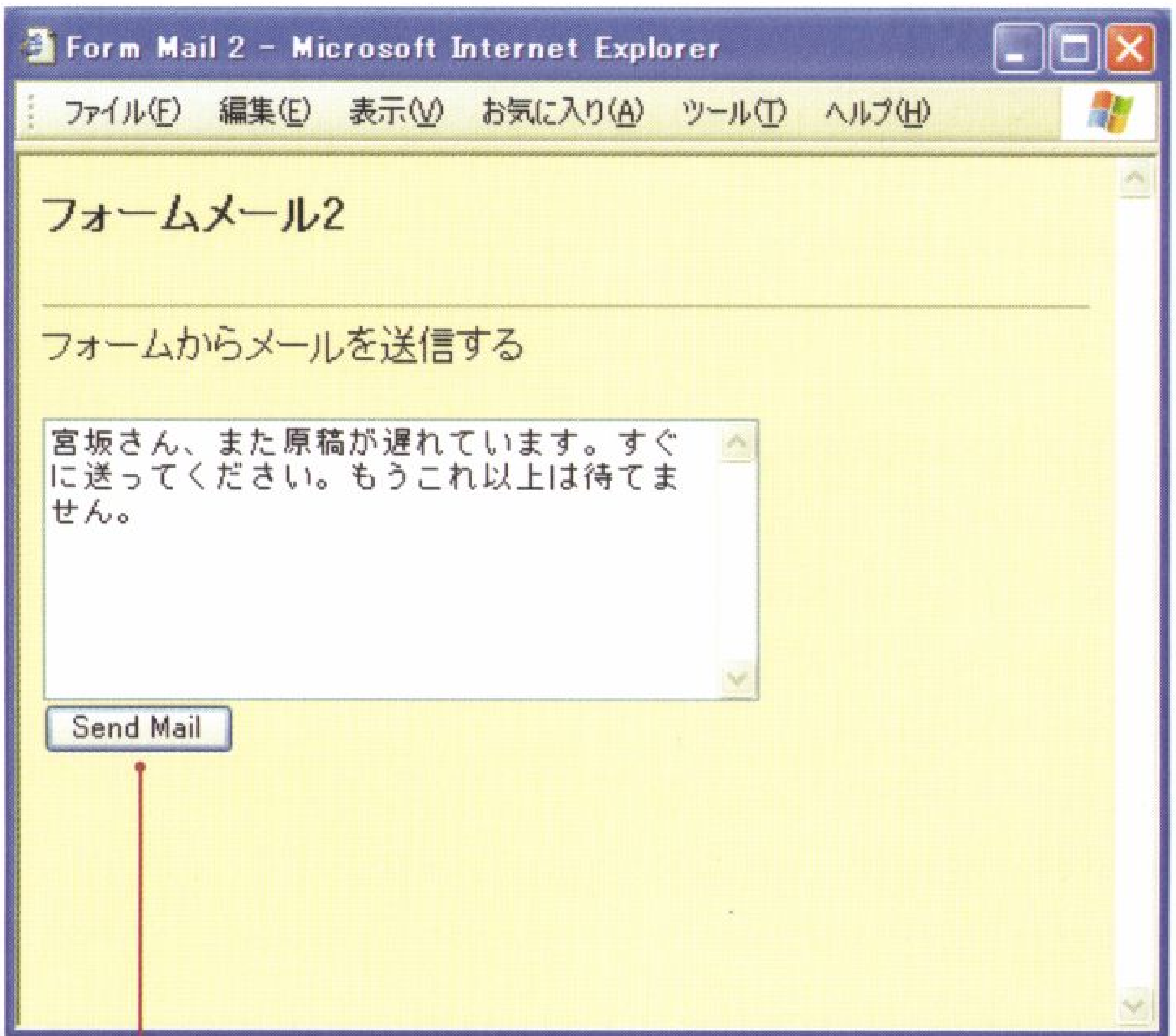
フォームメールにさまざまな情報を付加する

<input type= “hidden” name= “element名” value= “フォームメールに付加するテキスト” >

複数のWebページに違う内容のメールフォームを用意している場合、どの内容に対してメールを送ってきてくれているのかがわからなくなってしまいます。形式が特徴的だったらよいのですが、そうでない場合は、どの部分に対する感想や意見なのかもわからなくなってしまいます。そんな問題を避けるために、どこのWebページからメッセージが来ているのか、という情報をメール本文に加えてみます。

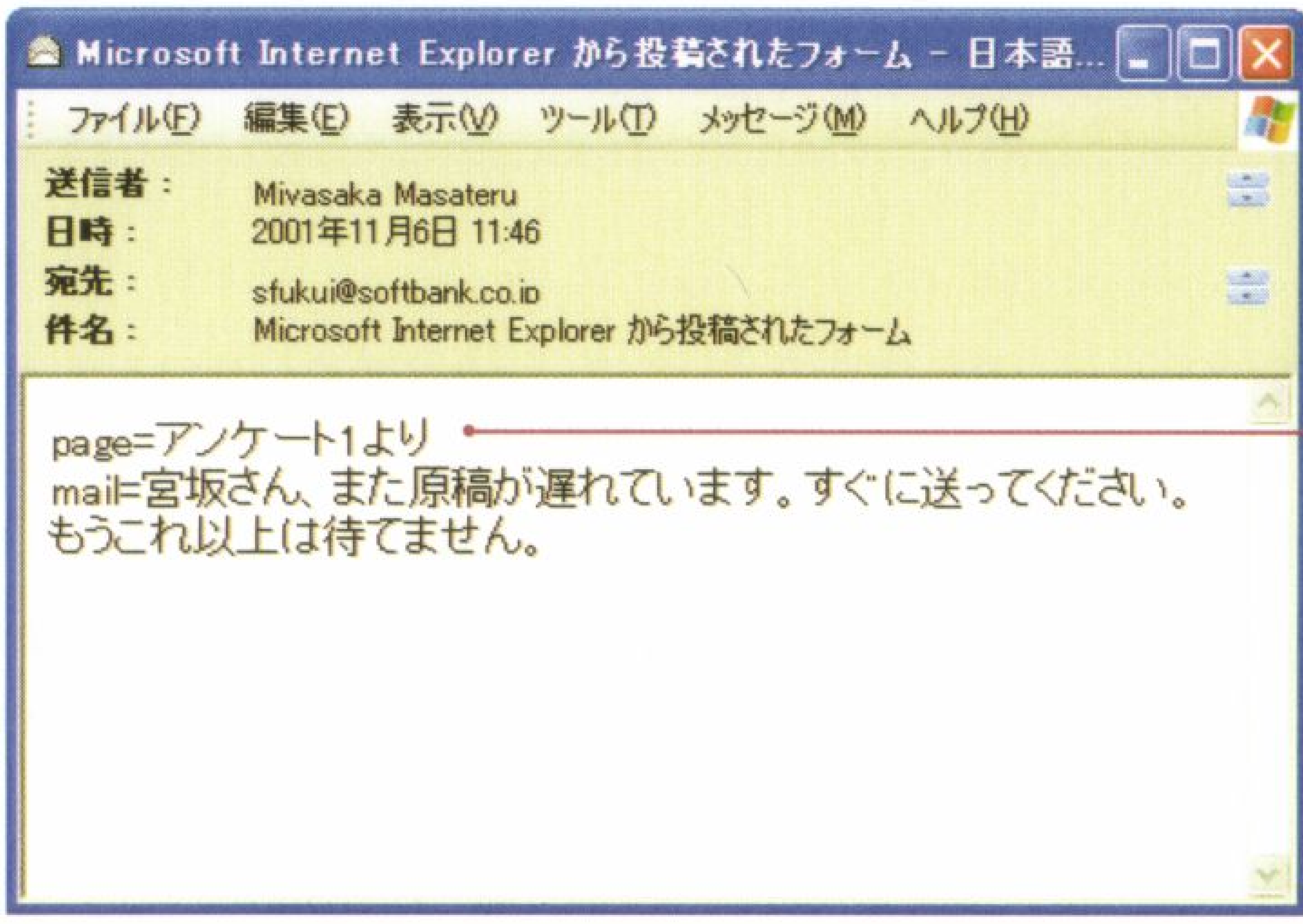
DISPLAY

▼送信するフォームメール



ここをクリックして送信

▼受信したメールソフトでの表示



メールにこの情報が
追加されている

SOURCE formmail2.html

```
<html>
<title>Form Mail 2</title>
<body bgcolor="#fff8dc">
<h3>フォームメール2</h3>
<hr />
フォームからメールを送信する<br />
<form name="form1" method="POST" action=
⇒ "mailto:address@domain"onsubmit="document.        ①
⇒ form1.encoding='text/plain'">
    <input type="hidden" name="page"
⇒      value="アンケート1より" />                  ②
    <textarea rows="8" cols="40" name="mail">         ③
    </textarea><br /><input type="submit" value="Send Mail">
</form>
</body>
</html>
```

DETAILS

このスクリプトでは、[Send Mail] ボタンをクリックすると、テキストエリアに入力した文字に、Webページの名前が付け足されたメールとして送信されます。

①の箇所で、フォーム全体には「form1」という名前が付けられています。action属性には送信先のメールアドレスを設定し、onsubmitイベントハンドラにエンコード方法を指定します。

このサンプルでは、通常のフォームに加えて、画面上に表示されない"hidden"というフォームを配置しています。このhiddenフォームには「page」という名前が付けられ、value属性の値として「アンケート1より」という文字列が設定されています（②の箇所）。また、文章を入力してもらうテキストエリアの部分には「mail」という名前が付けられています（③の箇所）。

Send Mail ボタンをクリックすると、<form>タグのaction属性で設定されているアドレスにメールが送信されます。このとき送信されるメールの内容は、以下のようになります。

```
page=アンケート1より
mail=テキストエリアに入力した文字列
```

つまり、画面上では見えていないhiddenフォームの内容も、本文に自動的に記入されるのです。

T I P S

hidden要素は、Webページをインタラクティブにするのに重要な役割を持っています。HTMLでは、記述した要素は表示されてしまうため、Webページ内で必要な情報はすべて表示しないといけないと思いがちですが、hidden要素を用いればWebページの裏？　に隠すことができます。記述は以下のようになります。

```
<input type="hidden" [name="名前"][value="値"] />
```

name属性とvalue属性はオプションですが、値を入れておくには両方ともないと意味がありません。しかし、後からスクリプトなどで操作することができ、添え字によるアクセスも可能なので、操作した結果を格納する入れものとして用意しておくということもできます。

hidden要素もフォームの1種なので、他のフォームの要素と同様にアクセスすることができます。name属性が指定されていれば名前による参照も、添え字による参照も可能です。

10-3 フォームメールにWebページのタイトルを付加する

WIN IE 4↑ N 4　MAC IE 4↑ N 4

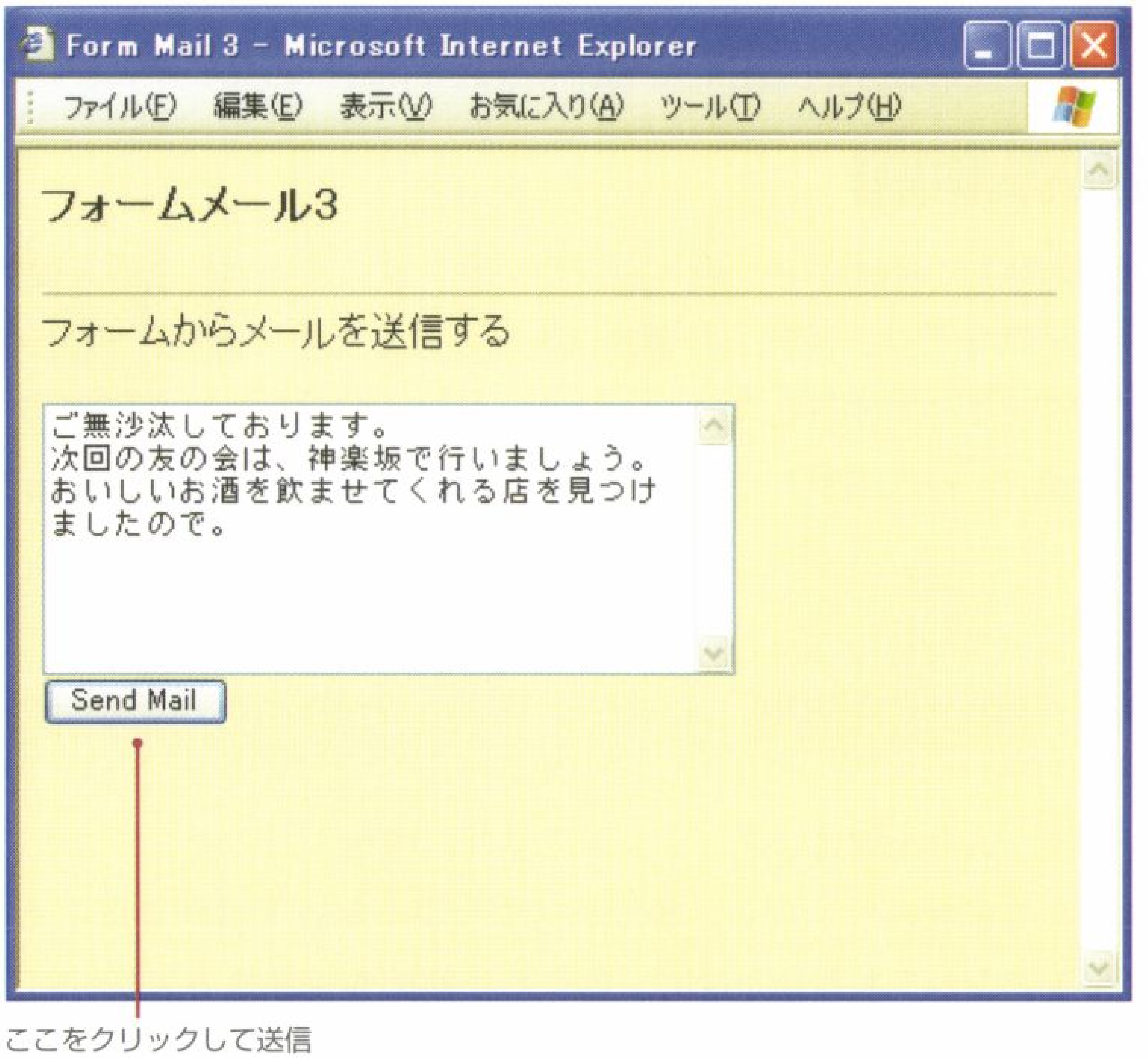
document.formオブジェクト名.elementオブジェクト名.
value=document.title

このサンプルでは、Webページからメールを送信する際に、本文の最後にWebページのタイトルを加えます。Webページのタイトルを加えることで、複数のWebページに送信フォームを設置している場合などに、管理がしやすくなります。

フォームを使ってメールを送る場合、スクリプト部分を<form>タグ内に書くことも可能ですが、それだけだと複雑な処理を行うことはできません。メンテナンスのことを考えても、スクリプト部分は独立させたほうがよいでしょう。このサンプルでは、スクリプト部分を分けて記述しています。

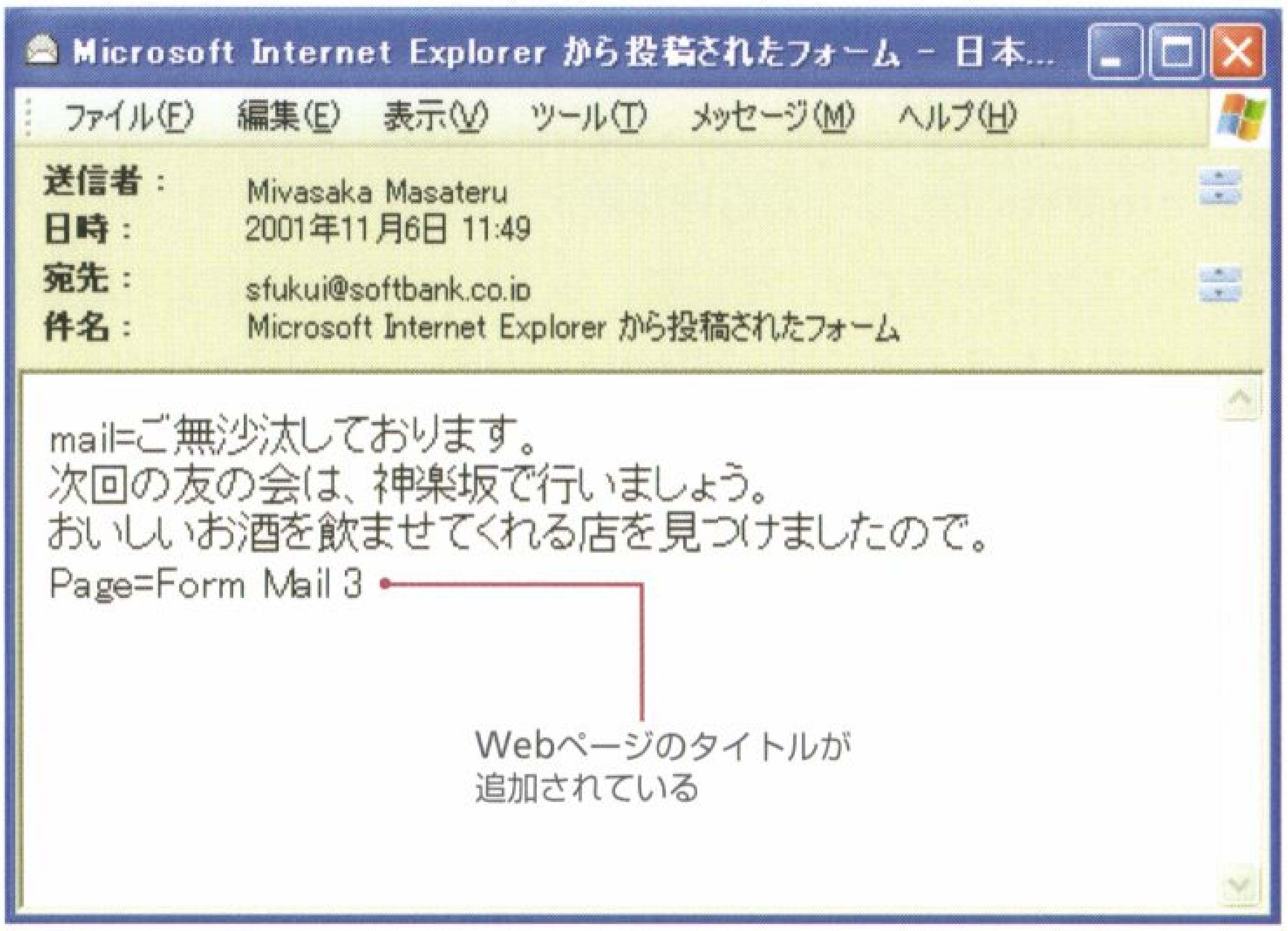
DISPLAY

▼送信するフォームメール



ここをクリックして送信

▼受信したメールソフトでの表示



SOURCE formmail3.html

```
<html>
<title>Form Mail 3</title>
<script language="javascript">
<!--
function addTitle(){
   document.sendmail.encoding='text/plain';                ②
   document.sendmail.Page.value=document.title;
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>フォームメール3</h3>
<hr />
フォームからメールを送信する<br />
<form name="sendmail" method="post" action="mailto:address
⇒ @domain" onsubmit="addTitle()">                          ①
   <textarea rows="8" cols="40" name="mail"></textarea><br />
   <input type="submit" value="Send Mail">
   <input type="hidden" name="Page">                       ③
</form>
</body>
</html>
```

DETAILS

このスクリプトでは、[Send Mail] ボタンをクリックすると（①の箇所）のonsubmitイベントハンドラによってaddTitle関数が呼び出されます。

②のaddTitle関数には、2つの処理が設定されています。1つはフォームで入力した文字列のエンコードです。そして、もう1つはメール本文の最後にWebページタイトルを記入することです。Webページのタイトルは<title>～</title>の間に記述した内容で、以下のように指定できます。

```
document.title
```

このサンプルでは次のように記述しています。

```
document.sendmail.Page.value=document.title;
```

「sendmail」は、フォーム要素に付けられている名前です。フォーム内には、③のところで「Page」という名前が付けられたhiddenフォームオブジェクトが配置されています。このオブジェクトのvalueプロパティに、Webページのタイトルを代入しているということです。

このフォームを送信すると、次のような形式で記述されたメールが<form>タグのaction属性に指定されているメールアドレスに届きます。

```
mail=テキストエリアに入力した文字列
Page=Webページのタイトル
```

10-4 WIN MAC

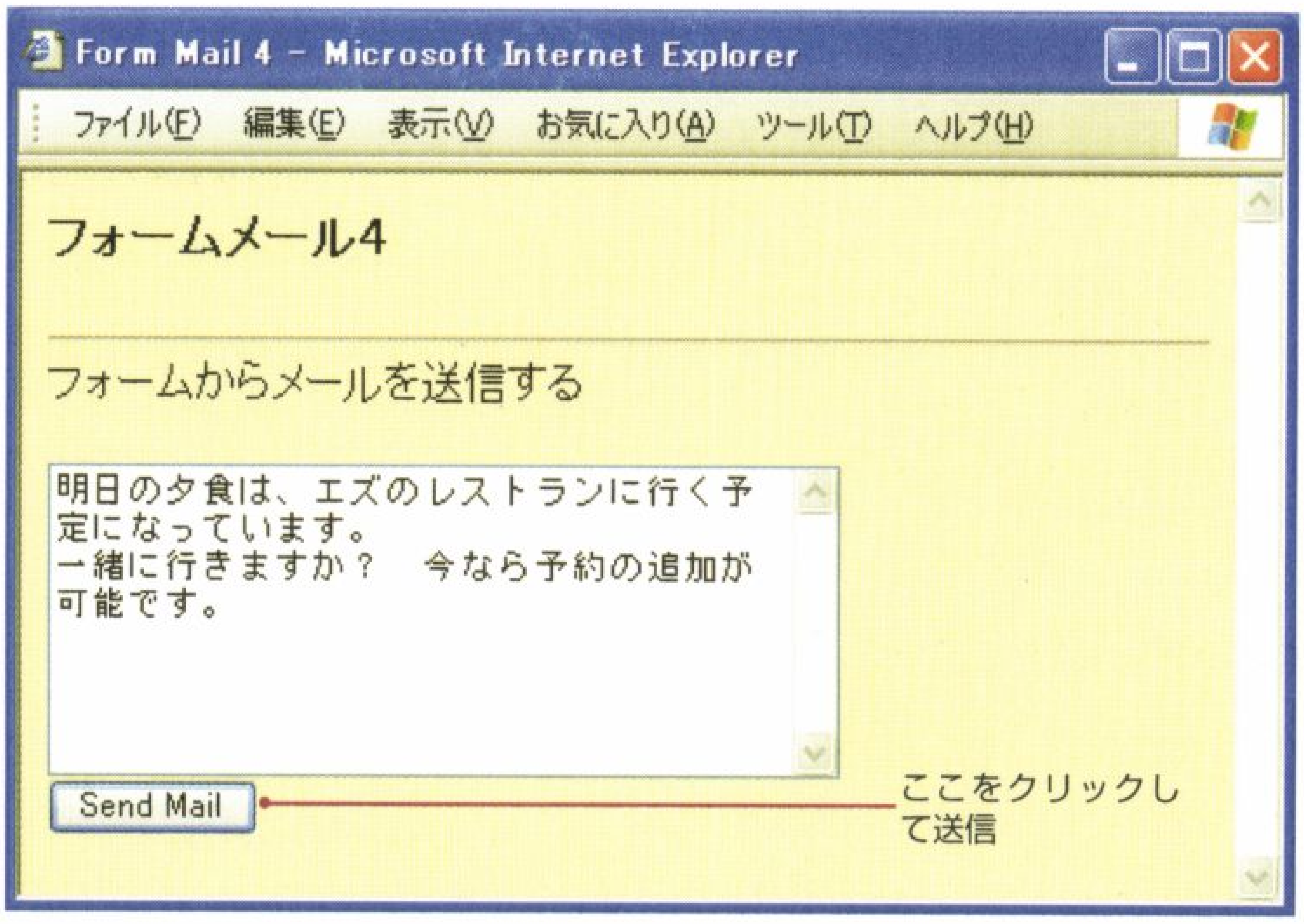
フォームメールにサブジェクトを設定する

action= "mailto:メールアドレス?Subject=サブジェクト"

メールを受信するとき、サブジェクトがわかりやすければ、大量のメールが届いても大事なメールを見落とすことはありません。また、サブジェクトごとにメールを振り分けられるメールソフトを使えば、内容ごとにメールを振り分けることも簡単にできます。

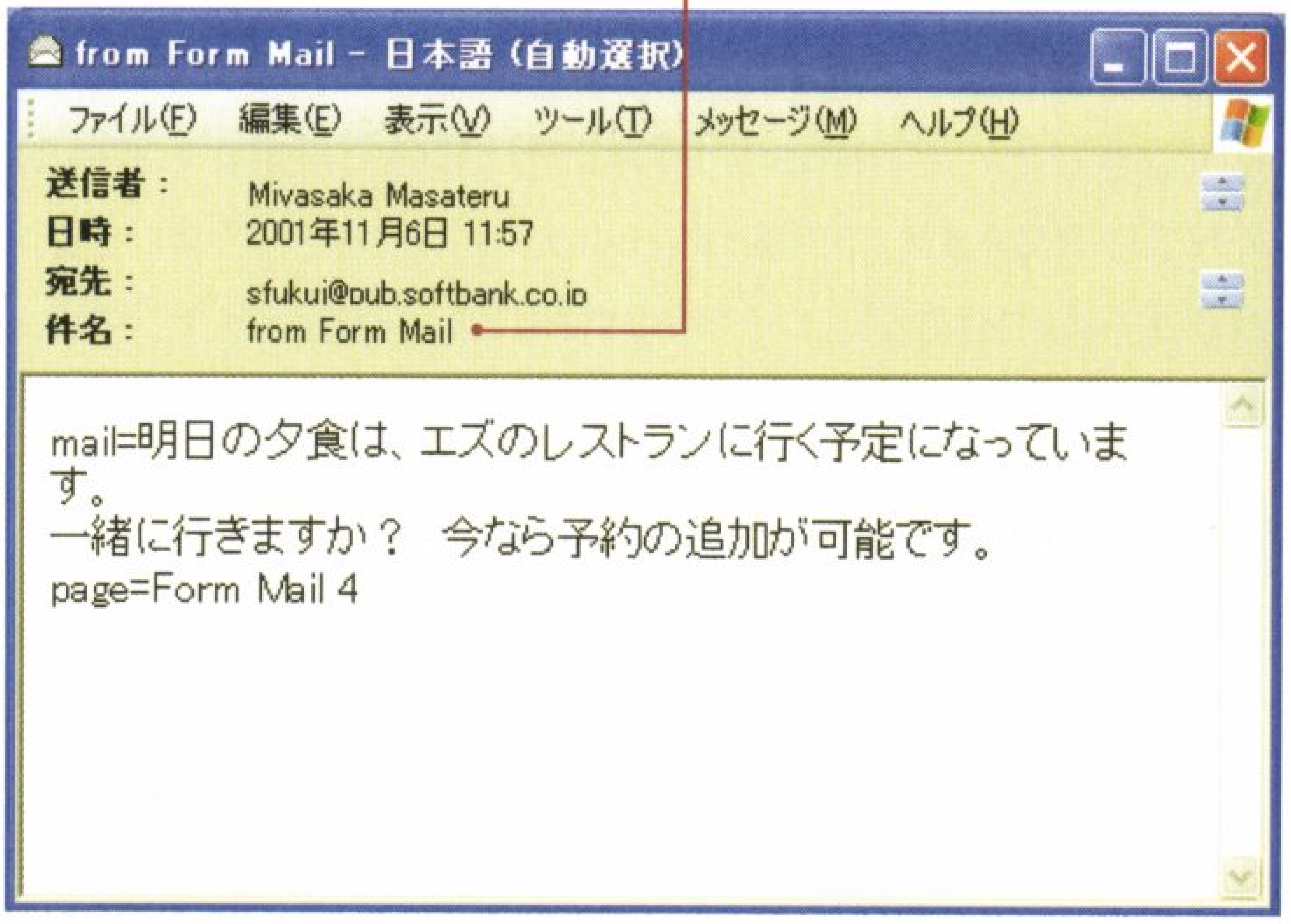
D I S P L A Y

▼送信するフォームメール



▼受信メールソフトでの表示

サブジェクト(件名)が付いている



SOURCE formmail4.html

```
<html>
<title>Form Mail 4</title>
<script language="javascript">
<!--
function addTitle(){
   document.sendmail.encoding='text/plain';
   document.sendmail.page.value=document.title;
}
//-->
</SCRIPT>
<body bgcolor="#fff8dc">
<h3>フォームメール4</h3>
<hr />
フォームからメールを送信する<br />
<form name="sendmail" method="post" action=
⇒ "mailto:address
⇒ @domain?subject=from Form Mail"
⇒ onsubmit="addTitle()">                                    ①
   <textarea rows="8" cols="40" name="mail"></textarea><br />
   <input type="submit" value="Send Mail" />
   <input type="hidden" name="page" value="アンケート1より" />
</form>
</body>
</html>
```

DETAILS

基本的な流れは、このセクションの他のスクリプトと同じです。[Send Mail]ボタンをクリックすると、<form>タグのaction属性で指定されているアドレスに、テキストエリアに入力した文章がメールとして送信されます。

送信されるメールにサブジェクトを付けるには、<form>タグのaction属性で設定するアドレス部分に、送信先とサブジェクトを指定します(①の箇所)。

```
action="mailto:メールアドレス?subject=サブジェクト"
```

つまり、アドレスの直後に「?」で区切って「subject=サブジェクト」と入力すると、設定した文字列が送信されるメールのサブジェクトになるのです。すぐに内

容がわかるようなサブジェクトを付けておいて、アンケートや感想などのフィードバックを効率よく利用しましょう。ただ、日本語のサブジェクトは文字化けする場合がありますので、なるべく英字にしておくのがよいでしょう。

COLUMN

XML

インターネットで注目されている技術の1つにXML（eXtensible Markup Language）があります。XMLはW3Cにより制定され、非常に柔軟な規格であるため、さまざまなマークアップ言語の定義に使われ、現在数多くの規格が検討、制定されています。

XMLを用いると、データ構造とスタイルを分離した文書データを作成できたり、システム間をつなぐメッセージを定義できたり、データの書式を定義できたりします。ここで重要なのは、これらのさまざまな仕様がXMLという一定の規格によって構築されていることです。これによりXMLの処理ができるならば、XMLに基づいた仕様のデータを一定のレベルまでは処理できることになります。この処理をするのがXMLパーサーと呼ばれるソフトで、OSに組み込まれたり、Webブラウザに組み込まれたりしています。

XMLパーサーはXML文書からデータの取り出し、挿入、データ内の操作、表示の作成などを行うことができます。このようにXMLパーサーがXML文書を扱うための機能を提供しているので、プログラマーはXMLパーサーの機能を用いればよいことになり、生産効率が上がります。

Webページ制作でのXMLとの関わり合いを考えてみると、HTMLをXMLベースで再構築したXHTMLがすでに実用段階に入っているのをはじめ、JavaScriptによりXMLパーサーを操作できる環境がすでに整っているなど、いつでも使える状態です。

このように、XMLはこれからのWebサービスの作成には欠かせない技術です。XMLについては、次のURLを参照してください。

URL▶ http://www.w3.org/TR/REC-xml

URL▶ http://www.fxis.co.jp/DMS/sgml/xml/wrapper-xml-19980210.html（日本語訳）

10-5 WIN e 4↑ N 4↑ MAC e 4↑ N 4↑

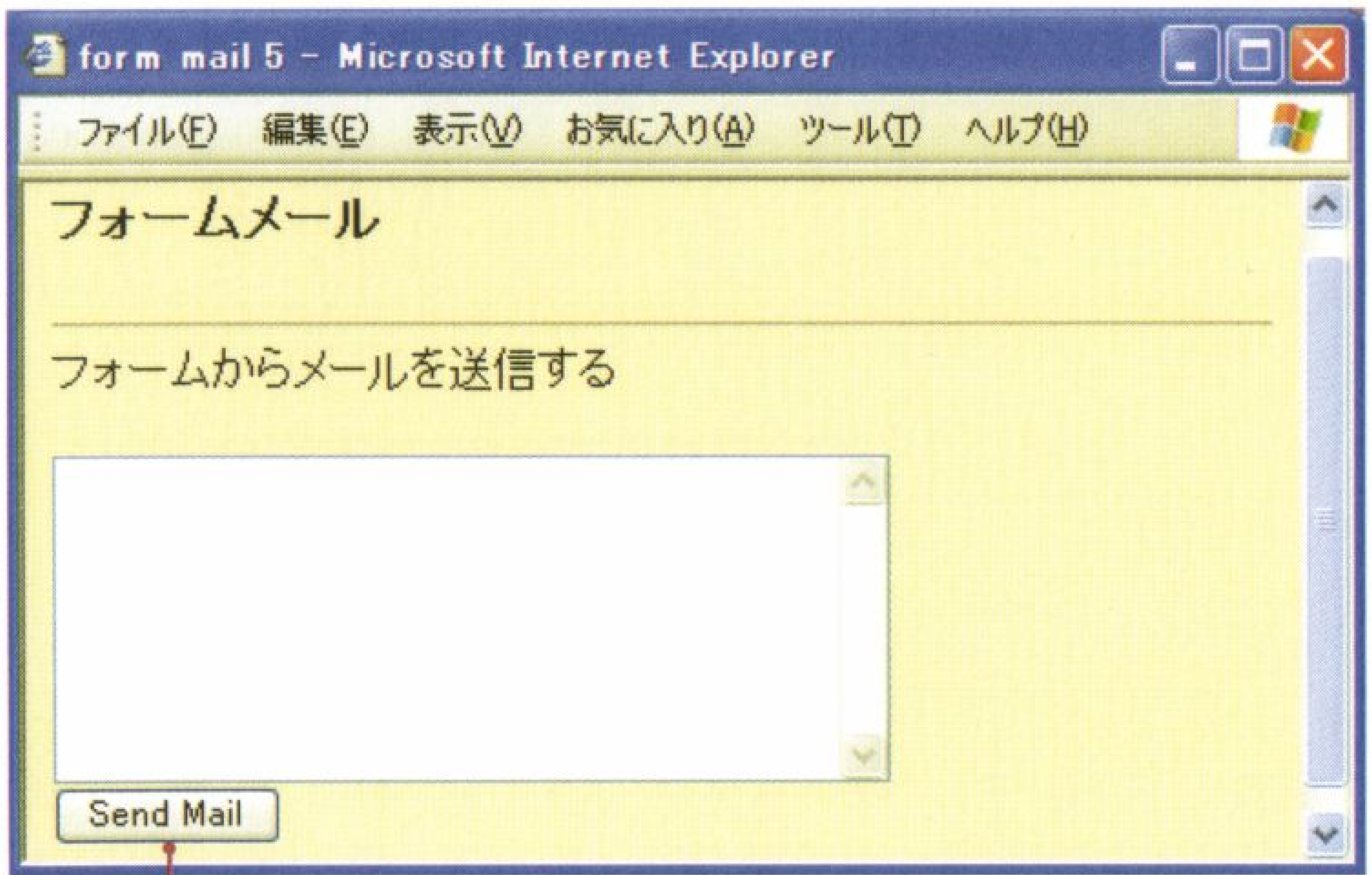
フォームを送信するときにチェックを行う

```
onsubmit=" return checkForm()
```

```
return false;
```

フォームを使ったメールを使うときに、ユーザーに必ず入力してほしい部分に入力が行われているかどうかのチェックを行いたいことがあります。もし、入力がされていないなどの場合は、メールの送信処理を途中で止めて、アラートを表示できるようにしておくと便利です。

D I S P L A Y



何も入力せずにここをクリックすると…



警告のアラートが表示される

SOURCE

```
<html>
<title>form mail 5</title>
<script language="javascript">
<!--
function checkForm(){
    if(document.sendmail.mail.value==""){
        alert("本文が入力されていません。");        ――②
        return false;
    }
    else{
        document.sendmail.encoding='text/plain';

        document.sendmail.page.value=document.title;
    }
}
//-->
</SCRIPT>
<body bgcolor="#fff8dc">
<h3>フォームメール</h3>
<hr />
フォームからメールを送信する<br />
<form name="sendmail" method="post"
⇒ action="mailto:address@domain?subject=from Form Mail"
⇒ onsubmit="return checkForm()">        ――①
    <textarea rows="8" cols="40" name="mail"></textarea><br />
    <input type="submit" value="Send Mail" />
    <input type="hidden" name="page" />
</form>
</body>
</html>
```

DETAILS

基本的な流れは、このセクションの他のスクリプトと同じです。異なる部分は、<form>タグに指定されているonsubmitイベントハンドラに、次の記述がされている点です（①の箇所）。

```
onsubmit="return 関数名()"
```

<form>タグのonsubmit属性でこのように関数の呼び出しを記述し、呼び出された関数からreturn命令で「false」が返ってくると、フォーム内容の送信が中止されます。onsubmitイベントハンドラで呼び出されているcheckForm関数では、フォームに入力がされているかどうかをvalueプロパティを使って調べています。もしも、入力がない場合には②の箇所のようにreturn命令を使って「false」を返します。このとき、呼び出し元のonsubmitイベントハンドラにも「false」が返されています。これによりフォームの送信を止めることになります。

フォームに入力がある場合には、タイトルを付加して送信します。

11-1 WIN 4↑ 4↑ MAC 4↑ 4↑

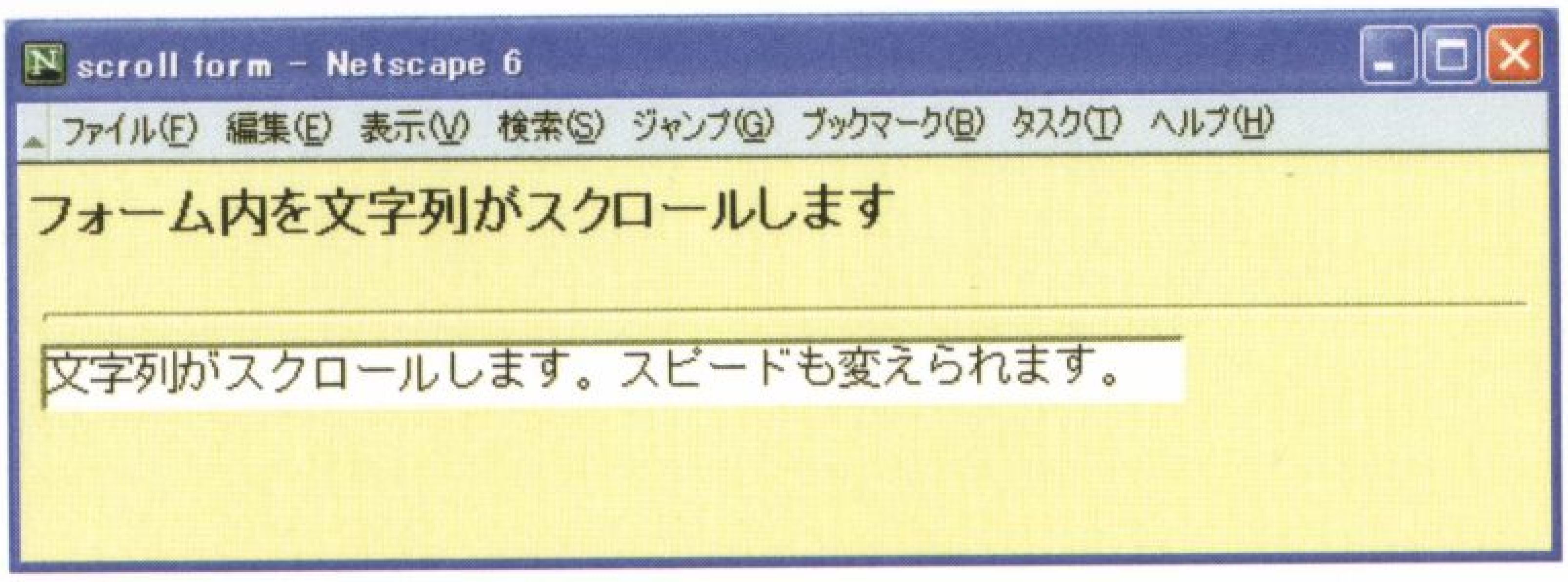
テキストボックスの文字列をスクロールする

setinterval ('実行する命令',命令を呼び出す間隔)

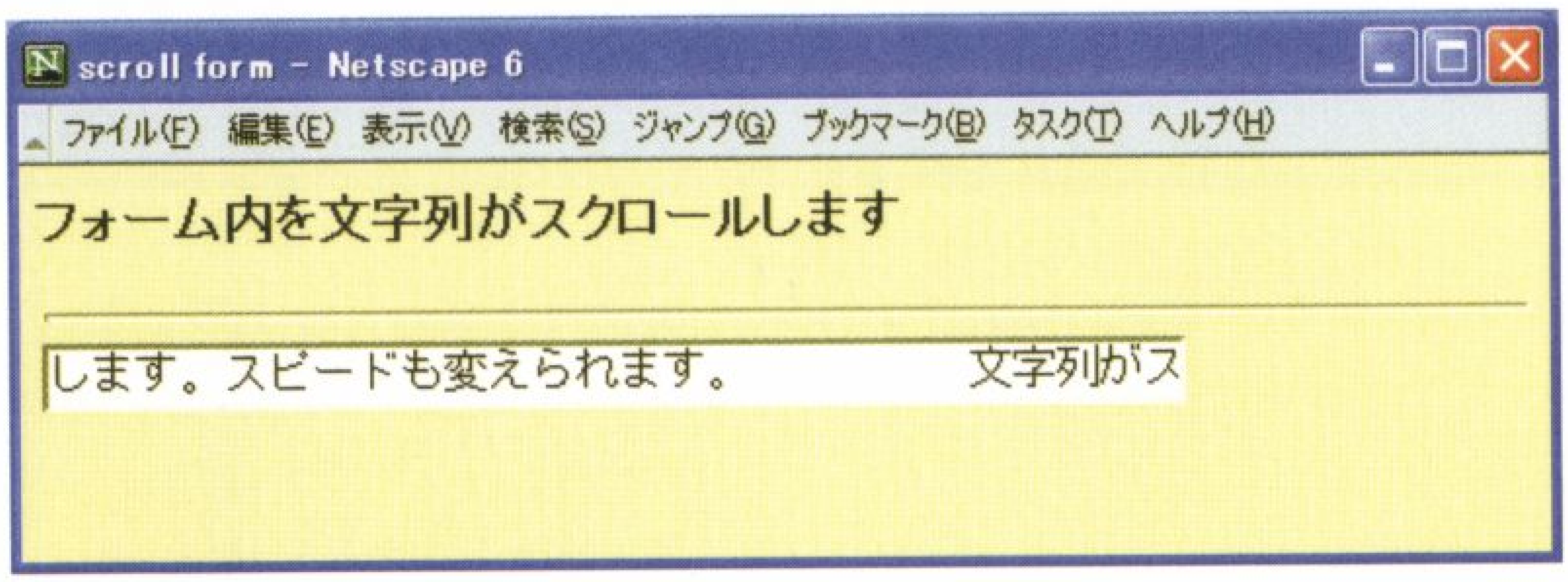
文字列.substring (抜き出す先頭位置,抜き出す終了位置)

テキストボックス内に、メッセージをスクロールさせるスクリプトです。ちょうど、電光掲示板のような感じになります。仕組みは簡単です。文字列を表示する位置を移動させるわけではなく、メッセージの先頭文字を切り取って、それを文字列の後ろにつなげるという動作を繰り返すだけです。

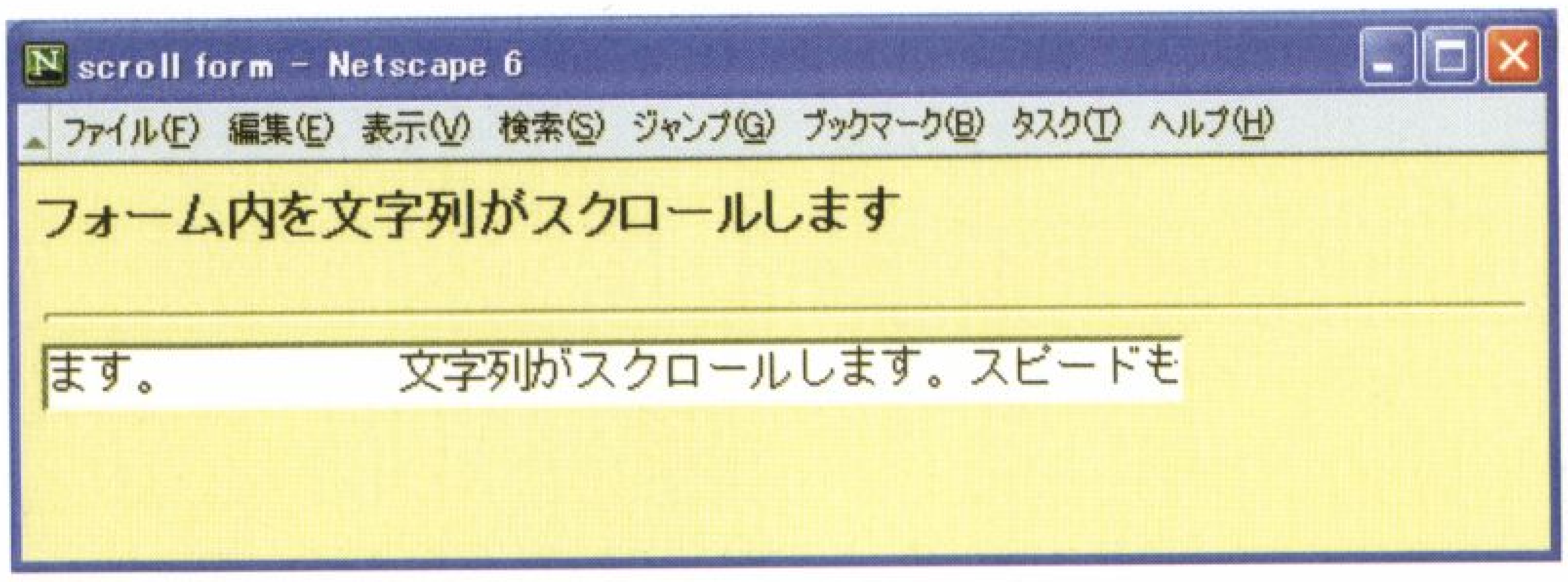
DISPLAY



↓



↓



ACCENT

SOURCE

scrolltext.html

```
<html>
<title>scroll form</title>
<script language="javascript">
<!--
msg="文字列がスクロールします。スピードも変えられます。";

function scrollText() {
    document.myForm.scroll.value = msg;  ――③
    msg = msg.substring(1,msg.length)+msg.substring(0,1);  ――④
}
//-->
</script>
<body onload="setInterval('scrollText()',200)"
➡   bgcolor="#fff8dc">  ――②
<h3>フォーム内を文字列がスクロールします</h3>
<hr />
<form name="myForm">
    <input type="text" name="scroll" size="50" />  ――①
</form>
</body>
</html>
```

ACCENT

DETAILS

このサンプルでは、「myForm」という名前のフォーム上に、「scroll」という名前を付けたテキストボックスを配置しています(①の箇所)。この中で、メッセージをスクロールさせます。

Webページが読み込まれると、<body>タグのonloadイベントハンドラによって、setIntervalメソッドが実行されます(②の箇所)。

```
setInterval('scrollText()',200)
```

これは、0.2秒(200ミリ秒)ごとにscrollText関数を呼び出すという処理です。setIntervalメソッドは、一定間隔ごとに特定の関数や命令を呼び出す場合に使用されるメソッドで、次のような書式になります。

```
setInterval('実行する命令', 命令を呼び出す間隔)
```

scrollText関数では、テキストボックスに変数msgに格納された文字列を表示させています（③の箇所）。

```
document.myForm.scroll.value = msg;
```

ここでは、「myForm」というフォームの「scroll」というテキストエリアの値（valueプロパティ）を、msgに格納されている値に設定しています。次に④の部分です。substringメソッドは、指定した文字列から部分的に文字を抜き取るメソッドで、以下のような書式になります。

```
文字列.substring(抜き出す先頭位置,抜き出す終了位置)
```

位置の指定は先頭が0になるので注意してください。このsubstringメソッドを使って変数msg内の文字列を操作することにより、あたかも文字列がスクロールしているかのような効果を生み出します。その方法は、文字列の先頭の一文字を、文字列の最後に移動する動作を繰り返すのです。

まず、次のような記述により変数msg内の文字列の2文字目から文字列の最後までを取り出します。

```
msg.substring(1,msg.length)
```

この後ろに、以下の記述で取り出された文字列の1文字目を連結します。

```
msg.substring(0,1)
```

つまり文字列の先頭文字を結合することで、2文字目が先頭になり、文字列全体が1文字ずつずれたような状態になります。

その文字列は変数msgに格納され、再びsetIntervalメソッドによって0.2秒後にscrollText関数が呼び出されます。今度は1文字分ずれた文字列が表示されるので、左に移動したように見えます。これが繰り返されることによって、文字列が右から左にスクロールしているように見えるのです。



スクロールさせる速さの変更には、2つの方法があります。まず、抜き出して最後に付け足す文字列を2バイト（全角の1文字）単位にする方法です。次のような記述になります。

```
msg = msg.substring(2,msg.length)+msg.substring(0,2);
```

また、関数を呼び出す間隔を変化させることによっても、さまざまにスピードを変えられます。

11-2

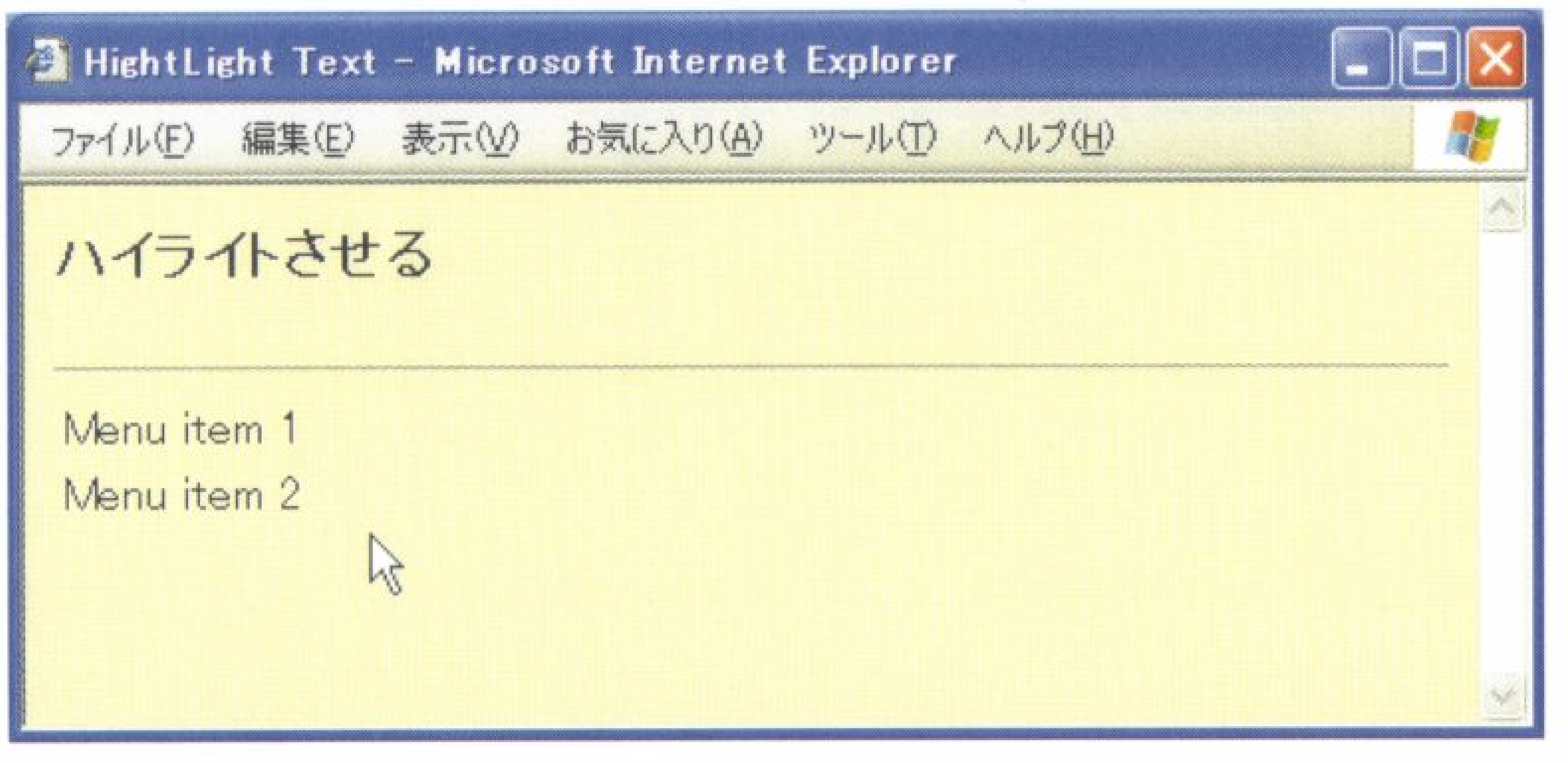
WIN IE 5↑ N 6↑　MAC IE 5↑ N 6↑

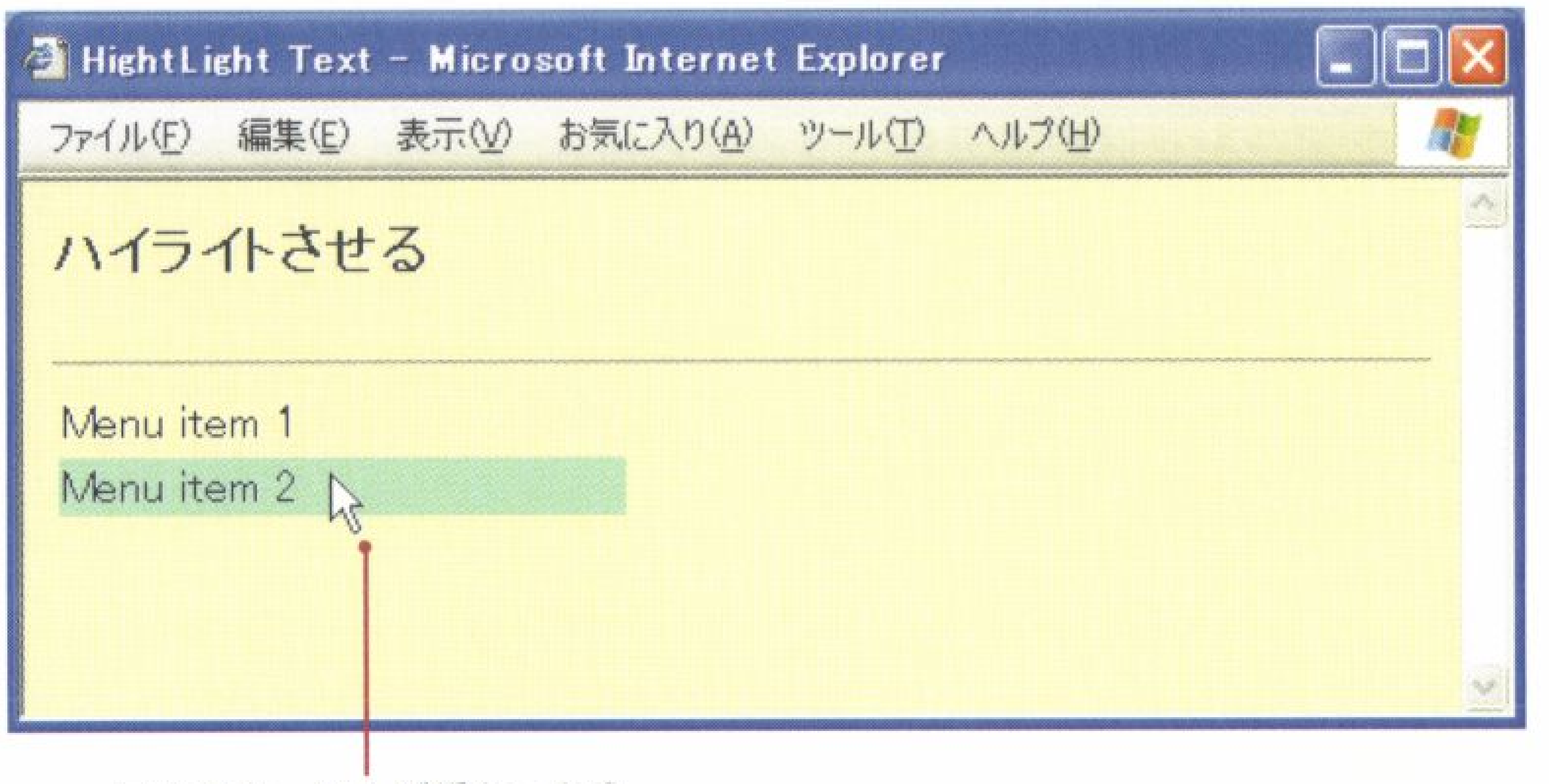
背景色をハイライトさせる

```
document.getElementById("エレメント名").style.
backgroundColor
```

マウスカーソルが何かにポイントしたとき、いろいろなイベントが起こると、ユーザーが非常にインタラクティブに感じるWebページになります。このようなWebページの実現にはイメージを切り替える方法もありますが、ポイントしたものの背景色を変えることでも実現できます。

DISPLAY





マウスカーソルが重なったら、
背景の色が変わる

ACCENT

SOURCE hightlighttext.html

```
<html>
<title>HightLight Text</title>
<body bgcolor="#fff8dc">
<script language="javascript">
<!--
function changeBgcolor(target, color){ ——————①
   document.getElementById(target).style.backgroundColor=color;
}
//-->
</script>
<h3>ハイライトさせる</h3>
<hr />
<table width="200">
<tr>
   <td id="str1" onmouseover="changeBgcolor('str1', '#00ffff')"
⇒      onmouseout="changeBgcolor('str1', '#fff8dc')">
      Menu item 1
   </td>
</tr>
<tr>
   <td id="str2" onmouseover="changeBgcolor('str2', '#00ffff')"
⇒      onmouseout="changeBgcolor('str2', '#fff8dc')">
      Menu item 2
   </td>
</tr>
</table>
</body>
</html>
```

DETAILS

このサンプルでは、<td>要素に囲まれた部分の文字列の背景色を、マウスのカーソルが重なったときに変更しています。また、マウスのカーソルが離れるときに、文字列の背景色を元に戻しています。

背景色を変更する部分は、①の部分のchangeBgcolor関数に記述されています。changeBgcolor関数は2つ引数を取ります。1つ目が対象となるHTML要

素の名前、2つ目が変更した後の背景色の色です。

このサンプルではIE5以降とNN6に特化しているため、getElementByIdメソッドを用いています。getElementByIdメソッドはDOM（Document Object Model）仕様に定義されているメソッドで、文章中からidを基にオブジェクトを返すメソッドです。getElementByIdメソッドは以下のような書式になります。

```
documentオブジェクト.getElementById(エレメント)
```

エレメントには、id属性に指定している属性の名前を指定します。

このメソッドにより返されるオブジェクトのスタイルを変更することで、背景色を変更しています。

背景色の変更は、背景色を表すプロパティに対してカラー名か16進のカラーコードを代入することにより行います。背景色のプロパティは、以下のように指定することができます。

```
document.getElementById("エレメント名").style.backgroundColor
```

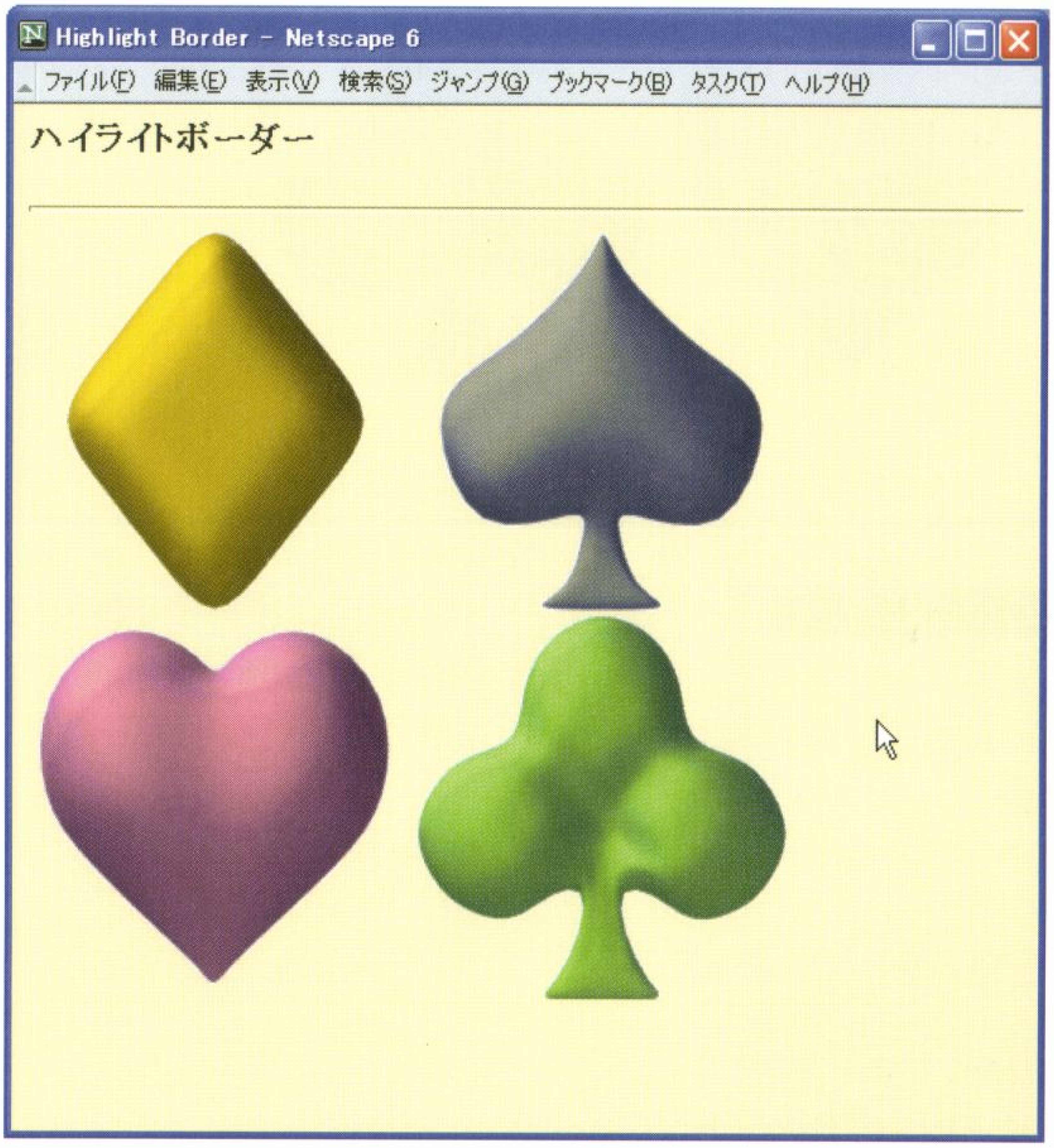
背景色を変更するのはカーソルを重ねたとき（onmouseoverイベントハンドラ）と、カーソルが離れたとき（onmouseoutイベントハンドラ）に行います。これらのイベントハンドラでchangeBgcolor関数を呼び出し、色を引数に与えることによりハイライトを実現しています。

枠線をハイライトさせる

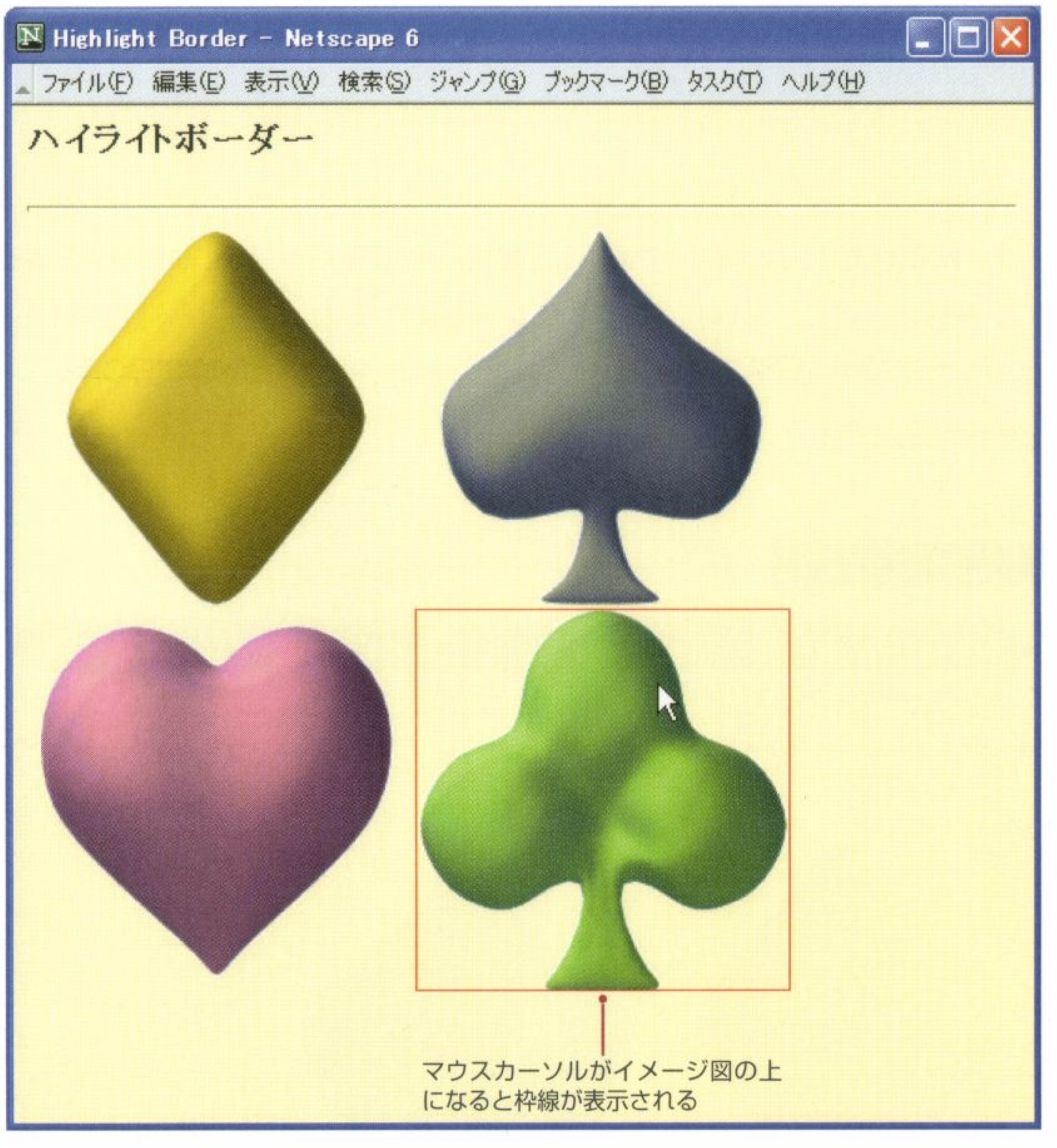
オブジェクト.borderColor=色

「11-2　背景色をハイライトさせる」で解説したように、マウスカーソルを何かにポイントしたときに、いろいろなイベントが起こると、ユーザーが非常にインタラクティブに感じるWebページを作ることができます。今度は、イメージの枠線を表示するようにしてみます。

DISPLAY



↓



SOURCE hightlightborder.html

```
<html>
<title>Highlight Border</title>
<style type="text/css">
   .borderimage{ border:1px solid white;}
</style>
<script language="javascript1.2">
<!--
function highlightBorder(which,color){
   if (document.all||document.getElementById){
      which.style.borderColor=color         ①
   }
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>ハイライトボーダー</h3>
```

ACCENT

```
<hr />
<img src="images/img1.gif" class="borderimage"
⇒   onmouseover="highlightBorder(this,'#ff0000')"
⇒   onmouseout="highlightBorder(this,'#fff8dc')" />
<img src="images/img2.gif" class="borderimage"
⇒   onmouseover="highlightBorder(this,'#ff0000')"
⇒   onmouseout="highlightBorder(this,'#fff8dc')" />
<br />
<img src="images/img3.gif" class="borderimage"
⇒   onmouseover="highlightBorder(this,'#ff0000')"
⇒   onmouseout="highlightBorder(this,'#fff8dc')" />
<img src="images/img4.gif" class="borderimage"
⇒   onmouseover="highlightBorder(this,'#ff0000')"
⇒   onmouseout="highlightBorder(this,'#fff8dc')" />
</body>
</html>
```

DETAILS

このスクリプトでは、マウスカーソルがイメージに重なったときに、イメージの周りに枠を出す処理を行っています。Webページ上には4つのイメージが配置されており、それぞれonmouseoverイベントハンドラと、onmouseoutイベントハンドラによりhighlightBorder関数が設定されています。そのため、マウスカーソルがイメージに重なったときと、マウスカーソルがイメージから外れたときにイベントが発生し、highlightBorder関数が呼び出されます。

highlightBorder関数は、2つの引数を必要とします。1つ目は枠線を変更するオブジェクト、2つ目は枠線の色です。highlightBorder関数では引数で与えられたオブジェクトの枠線を指定された色で変更します。そのため、呼び出し側でオブジェクトを指定する必要があります。

ここで指定しているのはthisというキーワードです。これは「このオブジェクト」という意味なので、呼び出したイメージを指定していることになります。

枠の色はborderColorプロパティに色を指定することにより変更することができます。そのため、受け取ったオブジェクト(which)の下のstyle.borderColorを変更するので、①のような記述になります。

onmouseoverイベントハンドラのときに赤(#ff0000)に変更し、onmouseoutイベントハンドラのときに背景色(#fff8dc)に変更しています。そのため、マウスカーソルが重なったときに枠線が見えるようになります。

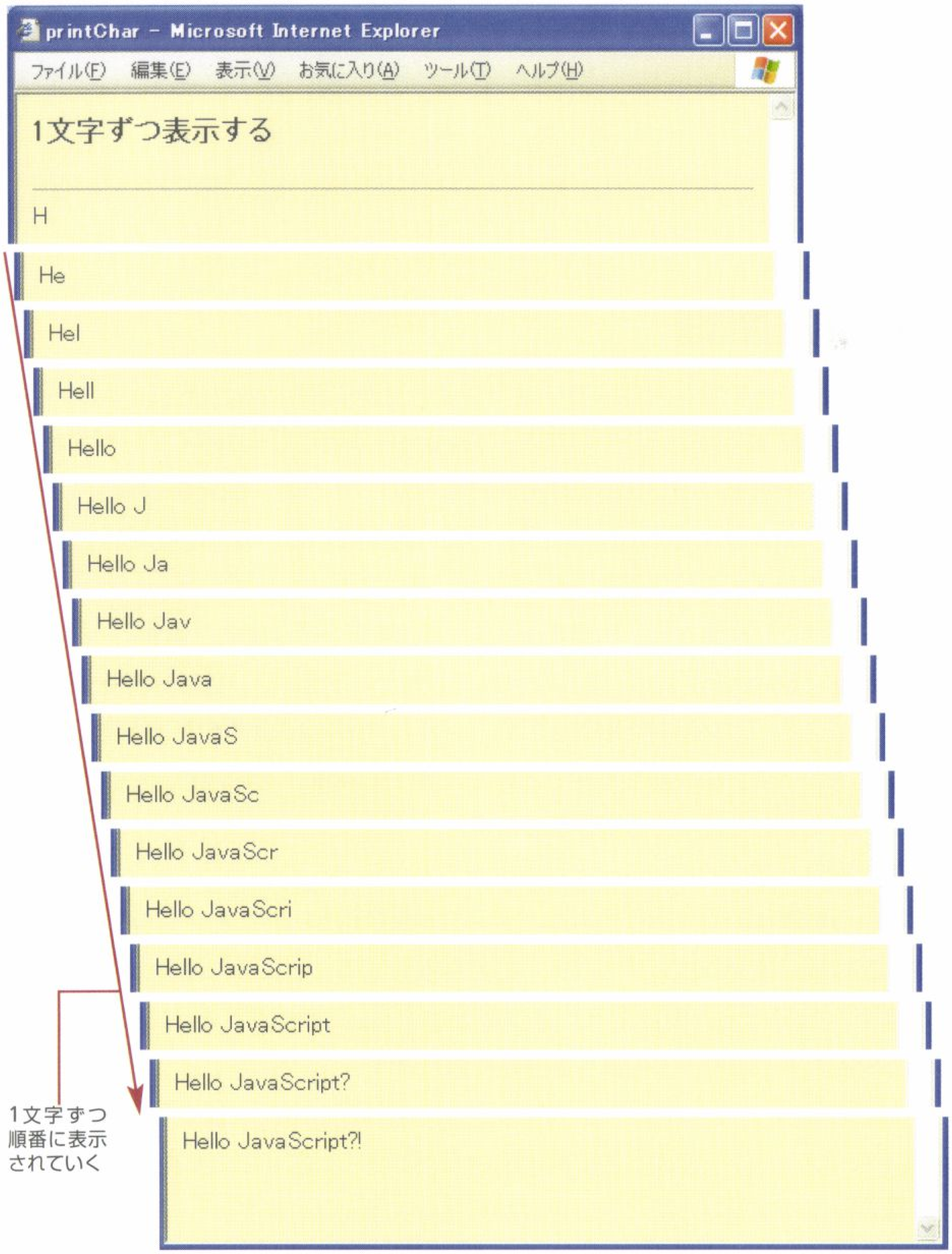
11-4 WIN IE 5↑ N 4↑ MAC IE 5↑ N 4↑

1文字ずつ表示する

timerID = setTimeout ("printChar()" , 呼び出す間隔)
document.layersオブジェクト
document.getElementByIdオブジェクト

1文字ずつ表示されていくというスクリプトです。タイプライタで入力しているような雰囲気を作り出すことができます。

DISPLAY



SOURCE printchar.html

```
<html>
<title>printChar</title>
<script language="javascript">
<!--
var i=0, printStr;
var msg = "Hello JavaScript?!";

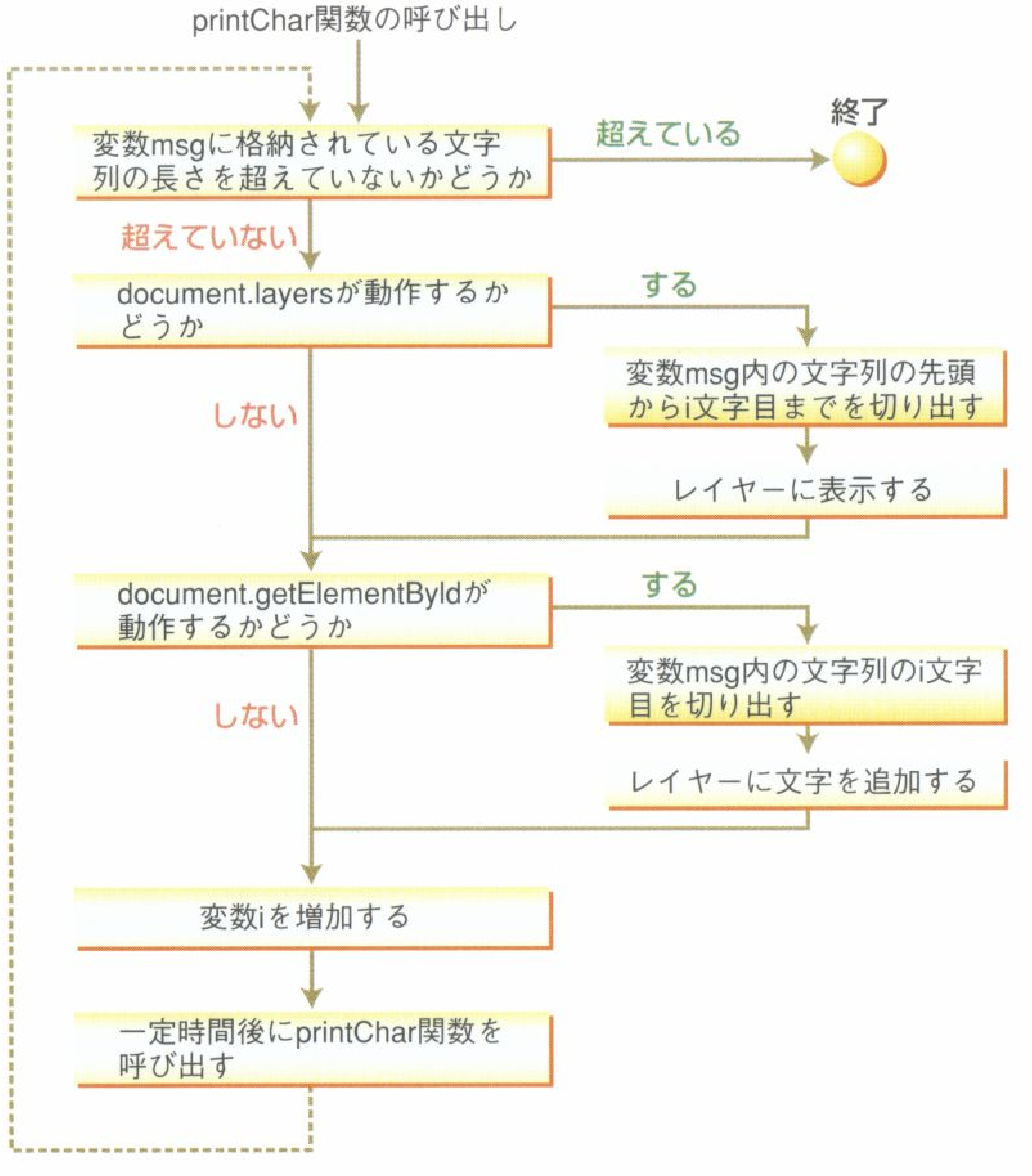
function printChar(){
   if( i>msg.length){                                    ①
      clearTimeout(timerID);
   }
   else{
      if(document.layers){                               ②
         printStr = msg.substring(0,i);
         document.sText.document.open();
         document.sText.document.write(printStr);
         document.sText.document.close();
      }
      else if(document.getElementById){                  ③
         printStr = msg.substring(i,i+1);
         var obj = document.createTextNode(printStr);
         document.getElementById("sText").appendChild(obj);
      }
      i++;
      timerID = setTimeout("printChar()", 200);          ④
   }
}
//-->
</Script>
<body onload="printChar()" bgcolor="#fff8dc">
<h3>1文字ずつ表示する</h3>
<hr />
<div id="sText" style="position:absolute"></Div>
</body>
</html>
```

ACCENT

DETAILS

　このサンプルでは、変数msgに格納された文字列を1文字ずつ表示します。スクリプトは、Webブラウザによって処理を変えるようにしています。NN4では表示するためのレイヤーを用意し、そこに1文字ずつ文字列を増やしていき、IE5以上とNN6では、レイヤーに対して、1文字ずつ追加していくことで実現しています。

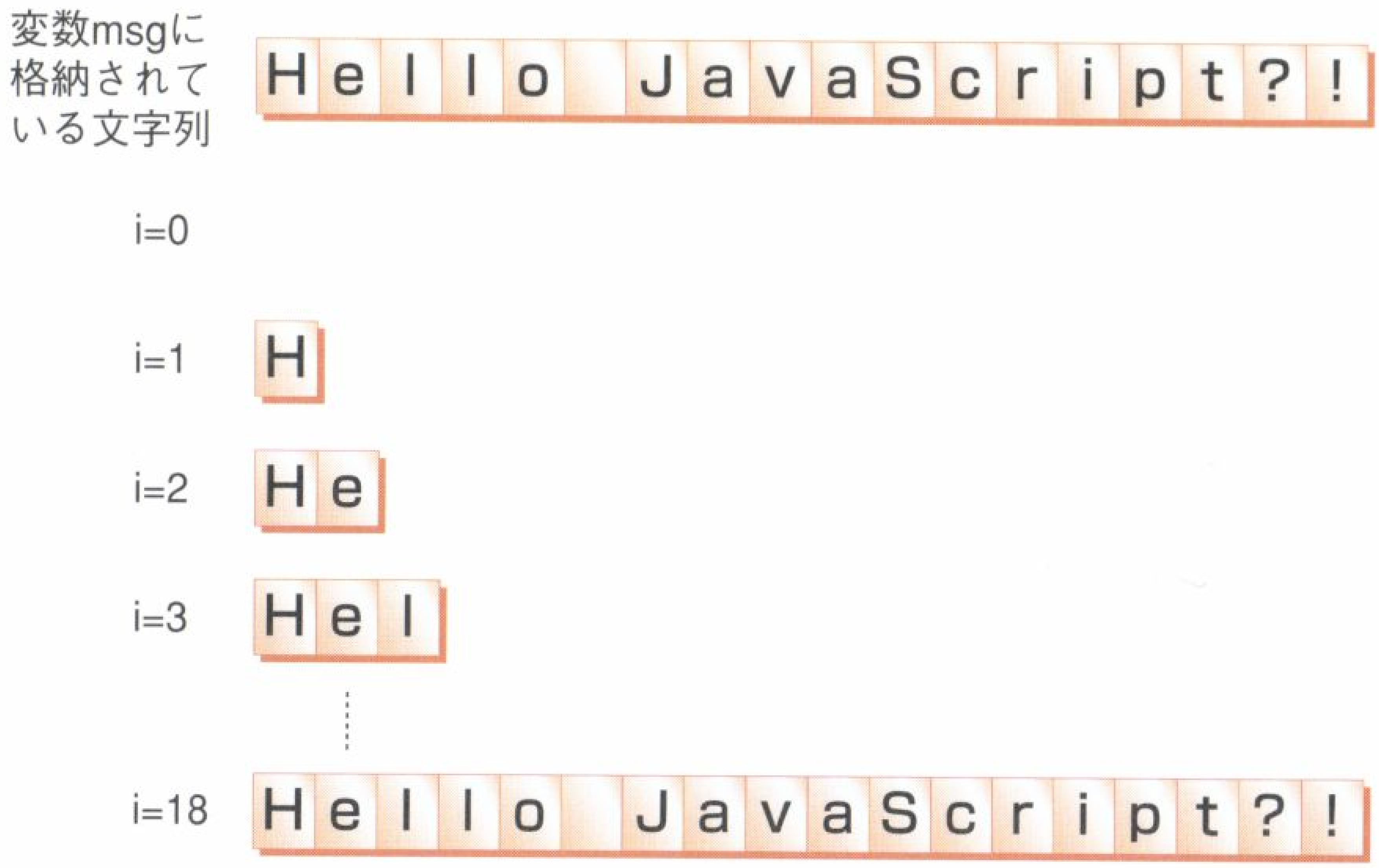
　1文字ずつ増やし、レイヤーに表示する機能はprintChar関数に実装されています。スクリプトのフローは、次のようになります。



　printChar関数では、変数msgの長さをlengthプロパティを用いて調べ、変数iがその長さを超えていないかどうか調べます（①の箇所）。変数iは、文字列の何文字目までを表示するかをコントロールする変数です。ここで超えている場合には、後述するタイマーによるprintChar関数の呼び出しをclearTimeoutメソッドで止めることによりスクリプト処理を終了しています。変数iが文字列の長さ

を超えていない場合には、文字列を表示する処理を行います。

NN4の処理は、②の部分のif文で真になった場合に行われる処理です。NN4かどうかを調べるには、document.layersオブジェクトが動作するかどうかで調べています。このオブジェクトはNN4でしか動作しないためです。このif文が真になった場合には、表示用の文字列の生成を行います。これは、変数msgから変数i文字分だけを切り出すという処理により行います。



変数iはだんだんと増えていくので、切り出される文字列が長くなっていきます。このようにして生成した文字列を、Webページ内のレイヤーに対して書き出すことにより、1文字ずつ表示する部分を実行します。

IE5以上とNN6を対象とする場合には、document.getElementByIdオブジェクトが動作するかどうかで調べます（③の箇所）。このif文が真になった場合の処理は、NN4のものとは異なっています。NN4では1文字ずつ増えた文字列を生成し、その文字列を表示していましたが、こちらの処理では、文字を1つずつ追加することにより行います。

このときにはcreateTextNodeメソッドにより、i番目の文字が格納されたノード（オブジェクト）を作成し、それをレイヤーの部分に追加するという実装をしています。ノードを追加するにはappendChildメソッドを用います。このような実装によりi番目の文字を追加します。

printChar関数自体は、1回呼び出されると変数iが1つ増えるので、1文字ずつ表示する文字列が増えることになります。これを連続的に増やすには、printChar関数の中で一定時間後に再度printChar関数を呼び出すようにセットしま

す。この呼び出しには、setTimeoutメソッドを用いています。setTimeoutメソッドは、指定時間後に1回だけ命令を呼び出す機能を持っています。

```
timerID = setTimeout("printChar()", 200);
```

スクリプトでは200ミリ秒ごとにprintChar関数を呼び出しています(④の箇所)。前述したように、タイマーをクリアするにはclearTimeoutメソッドを使います。このとき止めるタイマーを識別するために、setTimeoutメソッドで設定した変数timerIDを引数として与えます。

COLUMN

文字化けへの対処

JavaScript内でダイアログボックスやステータスバーに日本語を表示すると、文字化けをしてしまうことがあります。これは、日本の漢字コードが2バイトであることに起因します。Webブラウザは、うまく漢字を認識することができないのです。さらに日本語の文字コードにはJIS、Shift-JIS(SJIS)、EUC(Extended UNIX Code)の3種類があるため、いっそう問題が複雑化しています。

もしダイアログボックス内などに表示した文字が化けてしまった場合は、化けてしまう文字の前に「¥」を入れてみてください。たとえば、「表示」という文字を表示させたいときに「示」の字が化けてしまうとします。そのときは、「表¥示」としてみるのです。なお、環境によっては「¥」(円記号)は「\」(バックスラッシュ)と表示される場合があります。

なお、<meta>タグを用いると、自動的に日本語コードを切り替えてくれます。たとえば、次のように記述すると、JISで日本語エンコードします。

```
<meta http-equiv="Content-Type" content= "text/html;
⇒  charset=iso-2022-jp" />
```

これだけコンピュータが発達した現在でも、依然として日本語処理の問題は解決されていません。特にJavaScriptやCSSなどではいくつかの問題が報告されています。しかし、IE4以降ではそれらの問題はかなり解決されてきていますし、NN4.06以降でもUnicodeに対応したことで、徐々に環境は改善に向かっています。

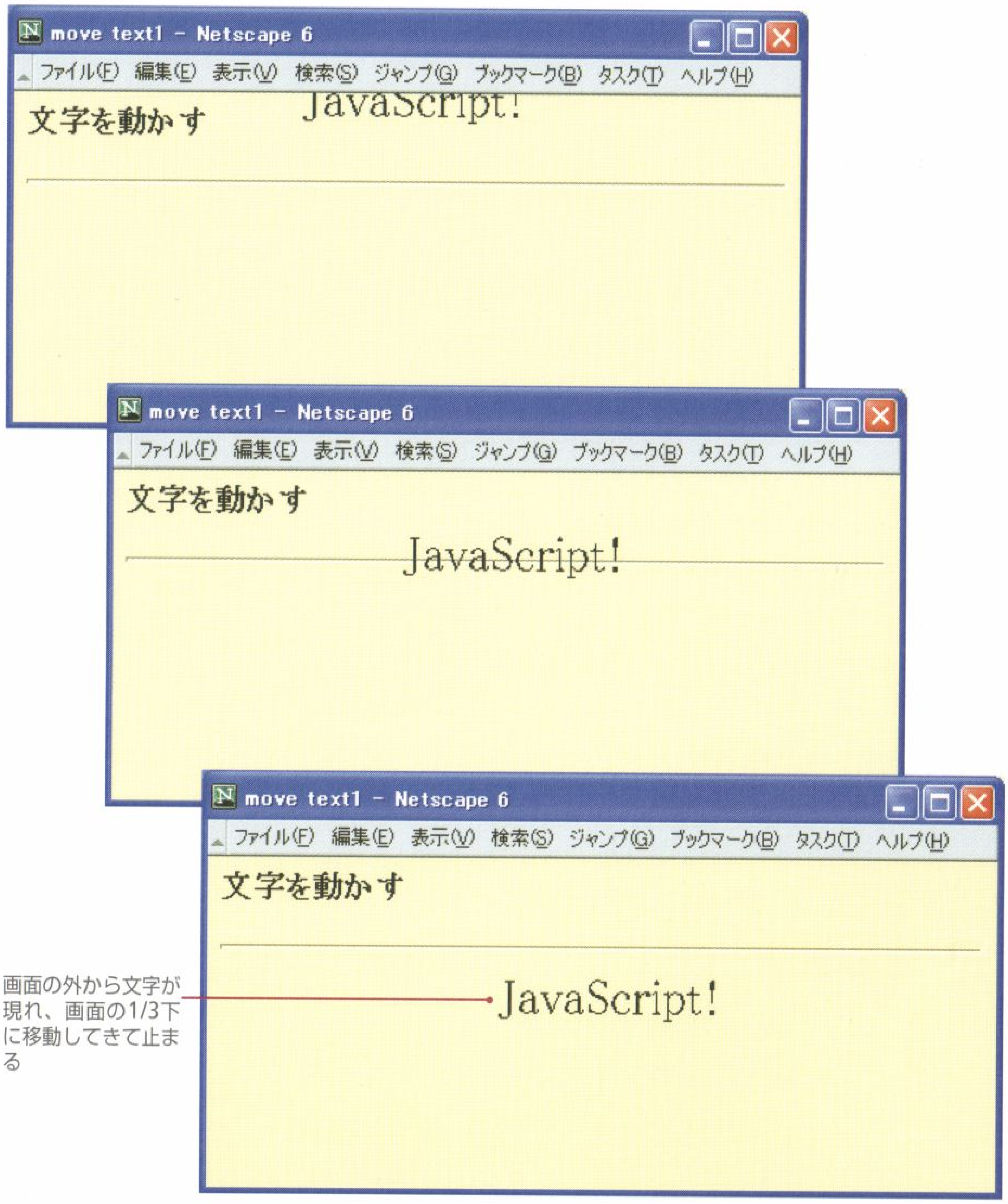
WIN MAC

文字を移動する

オブジェクト.style.top = Y座標
オブジェクト.style.left = X座標
オブジェクト.top = Y座標
オブジェクト.left = X座標

Webページに動きがあると、目を惹き付けるものがあり、Webページが華やかになります。アニメーションGIFなどで実現するという手もありますが、テキストを動かす程度ならスクリプトで実現することができます。

DISPLAY



画面の外から文字が現れ、画面の1/3下に移動してきて止まる

SOURCE movetext.html

```
<html>
<title>move text1</title>
<script language="javascript">
<!--
var start_y = -30; ──── ②
var step=2;
var last_y, posi_y, timerID;
var static_x = 200;
posi_y = start_y; ──── ③

function initPosition(){
    if(document.getElementById){ ──── ④
        if(document.all){
            last_y = document.body.clientHeight/3; ──── ⑤
        }
        else{
            last_y = innerHeight/3; ──── ⑥
        }
        document.getElementById("subtitle").style.top = start_y;
        document.getElementById("subtitle").style.left = static_x;
    }
    else if(document.layers){ ──── ⑦
        last_y = innerHeight/3;
        document.subtitle.top = start_y;
        document.subtitle.left = static_x;
    }
    moveText();
}
function moveText(){
    if( posi_y > last_y ){
        clearTimeout(timerID);
    }
    else{
        posi_y += step;
        if(document.getElementById){
            document.getElementById
➔               ("subtitle").style.top = posi_y; ──── ⑧
```

```
        }
else if(document.layers){
            document.subtitle.top = posi_y; ――⑨
        }
        timerID = setTimeout("moveText()", 10);
    }
}
//-->
</script>
<body onload="initPosition()" bgcolor="#fff8dc">
<h3>文字を動かす</h3>
<hr />
<div id="subtitle"
⇒   style="position:absolute;font-size:30px"> ――①
JavaScript!</div>
</body>
</html>
```

DETAILS

このスクリプトでは、Webページのロード時に上からテキストが現れ、Webブラウザの表示領域の1/3だけ落ちてきたら止まります。

移動していく文字部分は、<div>要素によりレイヤーを作成しています。id属性とstyle属性を宣言し、style属性には「position:absolute」を記述しています。これはNN4で<div>要素をレイヤーとして認識させるためです。また、文字を大きくするために「font-size:30px」と記述しています（①の箇所）。

スクリプト部分は大きく2つに分かれています。1つはWebページロード時に行われる初期設定部分、もう1つは実際に文字を動かす部分です。

Webページロード時には、各種の値の設定と、文字を移動させるということを行っています。値は、次のものを設定しています。

設定変数	内容
start_y	文字列のY座標（縦方向）の初期位置
last_y	文字列のY座標（縦方向）の最終位置
static_x	文字列のX座標位置（このスクリプトでは固定）
step	文字列を移動させる間隔
posi_y	文字列のY座標

変数start_yは、②の箇所のように最初「-30」に設定されているので、Webブラウザの表示領域外にあることになります。変数last_yは、Webページ表示領域の高さの1/3が入るようになっています。

Webページの表示領域はIEとNNでは格納しているプロパティが異なるので、initPosition関数内でそれぞれ取得しています。変数static_xは、今回は上から落ちてくるだけなので、固定で「200」としています。変数stepは、文字を移動させる間隔です。これを小さくすると文字が滑らかに移動します。また変数posi_yは、文字列のY座標を格納する変数です。初期位置は変数start_yと同じなので、③のように値を代入しています。

ここまでの動作で、表示領域の取得以外はWebページの読み込みの時点で行われています。

Webページがすべて読み込み終わったら、<body>要素のonloadイベントハンドラでinitPosition関数を呼び出します。initPosition関数では、文字列を含む<div>要素に対して値を与えています。<div>要素の位置の設定は、NN4の処理と、IE5以上、NN6の処理方法が異なりますので、ブラウザを判定して、それぞれ分けて記述しなければなりません。

IE5以上とNN6ではDOMが実装されているので、document.getElementByIdオブジェクトが動作します。そのため、④のif文が真になり実行されます。このスクリプトでは、Webブラウザの表示領域の高さを取得し、文字が動く量を決めています。しかしIEとNNでは、これらを取得するプロパティが異なるので、さらにdocument.allオブジェクトが動作するかどうかを調べています。IE5以上ではdocument.allオブジェクトが動作するので⑤の部分が、NN6では⑥の部分が実行されます。Webブラウザの表示領域が取得できた後で、それぞれX座標、Y座標を与えます。

NN4では、document.layersオブジェクトが動作するので、⑦のif文が真になり実行されます。NN4でもinnerHeightプロパティからWebブラウザの表示領域の高さを取得し、topプロパティ、leftプロパティを用いて文字列に初期状態の位置を与えています。

ここまでの初期設定がすべて終了すると、initPosition関数内でmoveText関数を呼び出します。moveText関数では、最初に文字列の位置がlast_yを超えていないかどうかチェックします。ここで超えている場合には、後述するタイマー(timerID)をclearTimeoutメソッドで解除します。

超えていなければ文字列の移動を行います。まず、最初に変数posi_yに変数stepを足し合わせ、それを文字列のY座標に代入します。ここでの文字列が格納されたオブジェクトは表現が異なるため、処理を分割しています。

IE5以上とNN6では、⑧の箇所のようにしてオブジェクトを指定しtopプロパ

ティを、NN4では⑨の箇所のようにオブジェクトを指定しtopプロパティでY座標を指定します。

最後に、この処理が繰り返し行われるようにsetTimeoutメソッドでタイマーを設定します。タイマーは後で解除することが可能なように、変数timerIDでタイマー識別を行います。タイマーでは10ミリ秒後にmoveText関数が呼び出されるようにしています。

T I P S

ここで紹介したサンプルでは縦方向に文字を動かしましたが、横方向に文字を動かすこともできます。左横から横方向に文字を動かしたい場合は、pixelTopプロパティの代わりにpixelLeftプロパティ、topプロパティの代わりにleftプロパティを用います。

また、記述するときには、X座標とY座標にも気を付けるようにしてください。テキストと画像と扱っているものは違いますが、「11-7　画像を移動する」で左横からの移動を扱っています。

▼文字が左横から移動する

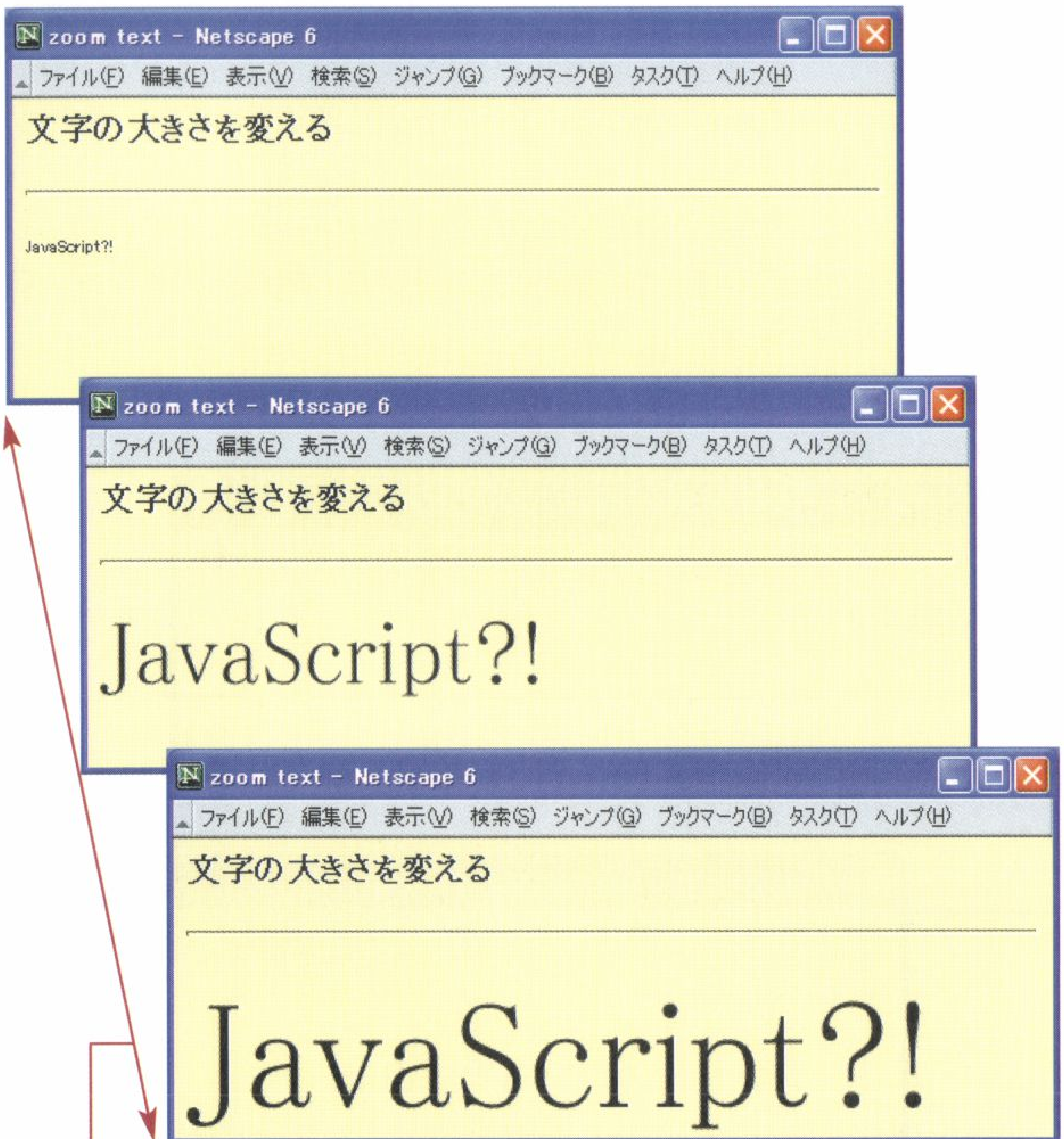


文字の拡大・縮小

オブジェクト.style.fontSize = フォントサイズ

JavaScriptを使うと、文字列の大きさを変更することもできます。あまりやりすぎるとセンスのよくないWebページになりますが、アクセントとして使うとよいかもしれません。このサンプルでは文字列の大きさの変え方を解説します。

DISPLAY



文字の拡大と縮小を繰り返す

SOURCE

zoomtext.html

```
<html>
<title>zoom text</title>
<script language="javascript">
<!--
var i, dstep, msg, msg2, timerID;
var step = 2;                                          ②
i=10;
dstep = step;

function zoomText(){
   if( i>90 ){
      dstep = -step;
   }                                                   ③
   if( i<10 ){
      dstep = step;
   }

   if(document.getElementById){                        ④
      msg1 = i + 'px';
      document.getElementById
⇥         ("subtitle").style.fontSize = msg1;
   }
   else if(document.layers){                           ⑤
      msg2 = '<div style="font-family:
⇥         Comic Sans MS; font-size:'+i+'px">JavaScript?!</div>';
      document["subtitle"].document.open("text/html");
      document["subtitle"].document.write(msg2);
      document["subtitle"].document.close();
   }

   i+=dstep;
   timerID = setTimeout("zoomText()",100);             ⑥
}
//-->
</script>
<body onload="zoomText()" bgcolor="#fff8dc>
<h3>文字の大きさを変える</h3>
<hr />
```

```
<div id="subtitle" style="position:absolute;top:80px">
➡   JavaScript?!</div>
</body>
</html>
```
①

DETAILS

　このサンプルでは「JavaScript?!」という文字列を周期的に大きくしたり、小さくしたりしています。

　今回のサンプルでは、IE5以上とNN6の場合はオブジェクトのフォントサイズを変更する方法で、NN4ではレイヤーを作成し、その内容を書き換えることにより実現しています。レイヤーは①の箇所のように<div>要素により生成しています。NN4でも操作できるように、id属性とstyle属性で「position:absolute」と宣言します。

　スクリプト読み込み時には、②の部分で以下の変数を設定しています。

設定変数	内容
i	文字列の大きさ
step	文字列の変化量
dstep	文字列の変化量（正負含む）

　変数 i は、実際に文字列の大きさを表しています。変数stepは、文字列が大きくなったり小さくなったりするときの変化の量です。今回は2ピクセルごとに大きくしたり小さくしたりするので「2」になっています。変数dstepは変数stepについて正負を考慮したものです。大きくなるときには変数dstepは正に、小さくなるときには変数dstepは負になります。

　文字列の大きさを変える部分は、zoomText関数に実装されています。zoomText関数では、まず文字の大きさを表す変数iの大きさを確認します。今回のサンプルでは10ピクセルから90ピクセルまで、2ピクセルごとに大きさが変化していきます。ここで変数 i が90を超えたら、変数dstepを負にするために、変数stepにマイナスを付けて代入します。また変数 i が10を下回ったら、変数dstepを正にするために、変数stepを代入します（③の箇所）。

　この後は、IE5以上とNN6、NN4の処理は異なります。IE5以上とNN6ではdocument.ElementByIdオブジェクトが動作するので、④の箇所のif文の中に記述された部分が動作します。この中では、変数iに「px」という文字列を連結し

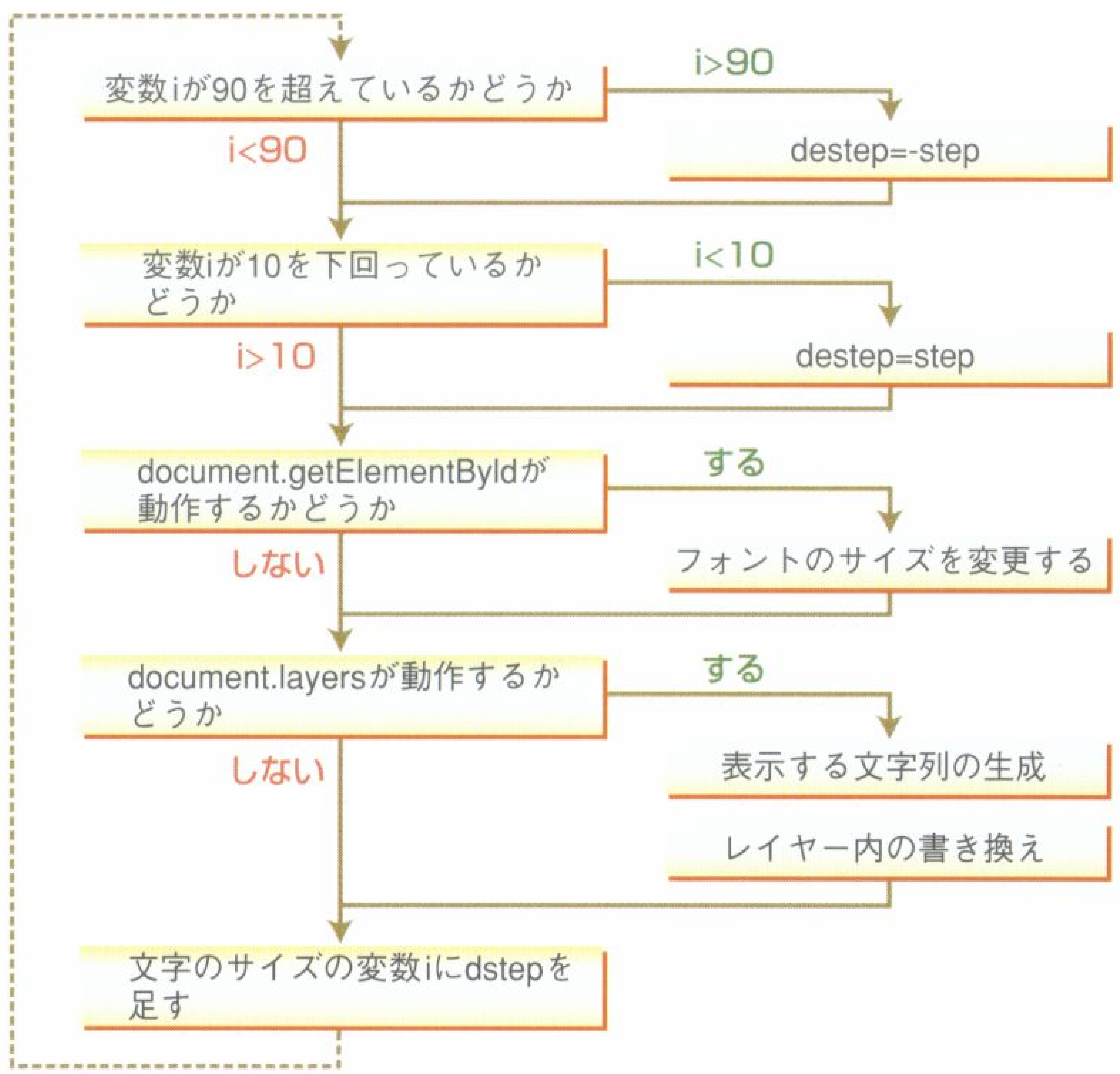
たものを変数msgに与えています。この変数をfontSizeプロパティに代入することにより、フォントサイズを変更しています。

NN4ではレイヤー内の文字列を変更することで行います。NN4ではdocument.layersオブジェクトが動作するので、⑤の箇所の処理が実行されます。ここでは<div>要素を用い、さらにサイズ指定はスタイルシートにより行っています。フォントのサイズを指定しているのはfont-size属性です。今回は滑らかに文字列の大きさを変えるためにスタイルシートを用いていますが、<font>要素を用いてサイズを指定してもかまいません。このように文字列を作成し、その後レイヤーのドキュメントを開き、writeメソッドで生成した文字列を記述します。この処理が終わると文字列が書き直されます。

それぞれ、文字のサイズを変更した後は、変数iに変数dstepを足し合わせて、次の処理に備えます。次の処理は⑥の箇所のようにsetTimeoutメソッドにより、100ミリ秒後に再度zoomText関数が呼び出されるようにしています。こうして繰り返しzoomText関数を呼び出すことにより、連続的に文字列の大きさが変わっていきます。

なお、Macintosh版IEでは、fontSizeがサポートされていないため、動作しません。

▼サンプルスクリプトの処理の流れ

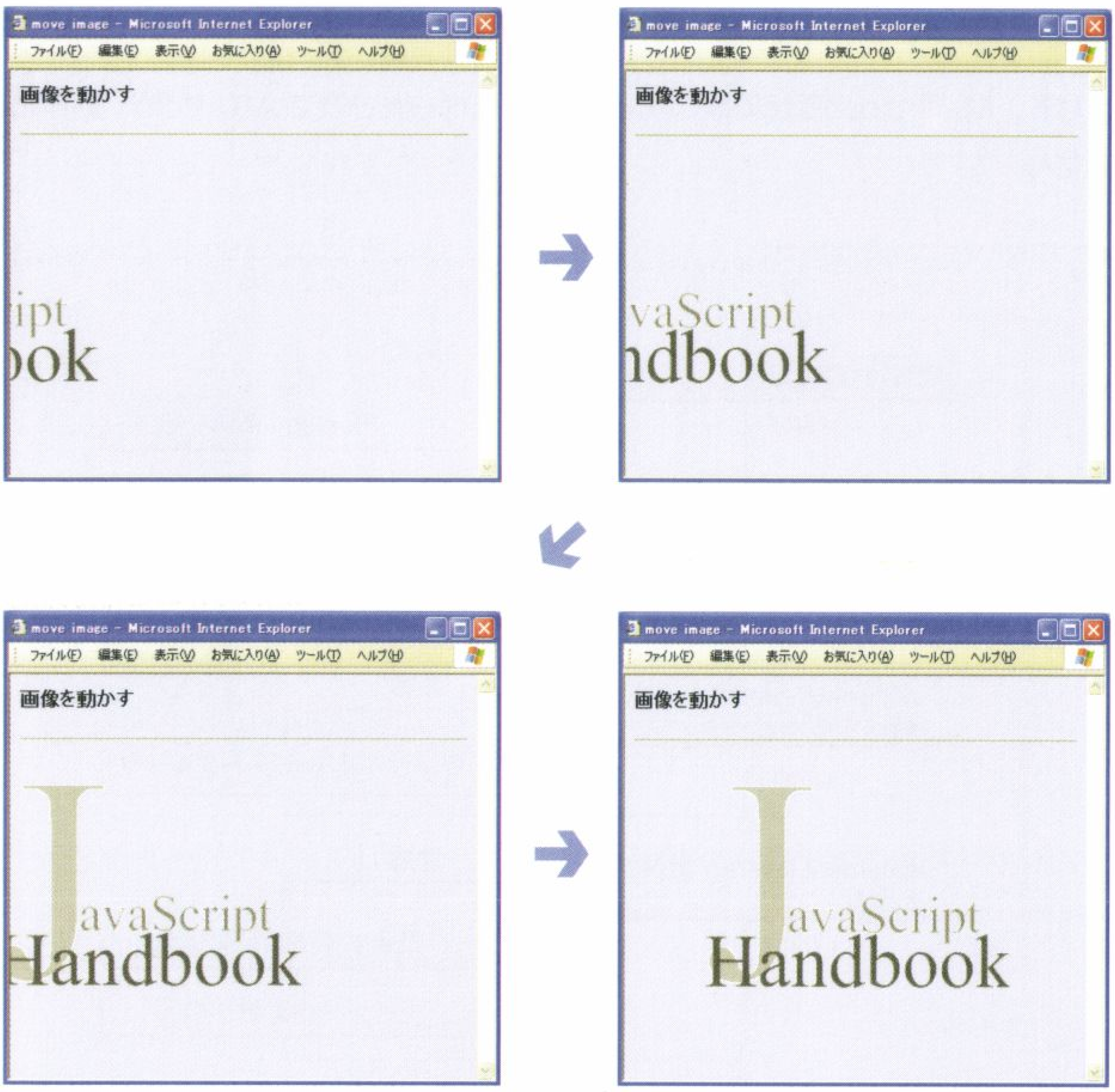


画像を移動する

オブジェクト.style.left = Y座標
オブジェクト.style.top = X座標
オブジェクト.left = Y座標
オブジェクト.top = X座標

Webページに動きがあると、それだけで華やかに感じるものです。文字列を動かすサンプルは「11-5　文字を移動する」ですでに紹介しました。同じように、画像を移動させることもできます。

DISPLAY



SOURCE moveimage.html

```
<html>
<title>move image</title>
<script language="javascript">
<!--
var start_x, static_y, step, last_x, posi_x, timerID;

function initPosition(){
    start_x = -300;
    static_y = 100;
    step=2;
    posi_x = start_x;
    if(document.getElementById){
        if(document.all){
            last_x = document.body.clientWidth/4;
        }
        else{
            last_x = innerWidth/4;
        }
        document.getElementById("subtitle").style.left = start_x;
        document.getElementById("subtitle").style.top = static_y;
        document.getElementById("subtitle").style.visibility =
⇒           'visible';
    }
    else if(document.layers){
        last_x = innerWidth/4;
        document.subtitle.left = start_x;
        document.subtitle.top = static_y;
        document.layers["subtitle"].visibility = 'visible';
    }
    moveImage();
}

function moveImage(){
    if( posi_x > last_x ){
        clearTimeout(timerID);
    }
    else{
        posi_x += step;
```

```
        if(document.getElementById){
            document.getElementById("subtitle").style.left =
⇒               posi_x;
        }
        else if(document.layers){
            document.subtitle.left = posi_x;
        }
        timerID = setTimeout("moveImage()", 10);
    }
}
//-->
</script>
<body onload="initPosition()" bgcolor="#e6e6fa">
<h3>画像を動かす</h3>
<hr />
<div id="subtitle"
⇒ style="position:absolute;visibility:hidden">     ─①
<img src="images/logo.gif" width="300" height="200" />
</div>
</body>
</html>
```


DETAILS

このスクリプトでは、Webブラウザの左横から画像を移動させます。スクリプト自体は、「11-5　文字を移動する」で紹介している文字を動かすスクリプトとほとんど同じで、画像を扱うためにいくつかの修正を加えています。このスクリプトでも、IE5以上とNN6、NN4で処理を分離しています。

まず画像ですが、<div>要素でレイヤーを作って、その中に画像を配置しています。これでレイヤーを操作すると画像もいっしょに動きます。IE5以上やNN6では直接イメージのオブジェクトの位置を操作できるのですが、NN4では動作しないため、画像をレイヤー内に配置し、レイヤーを動かすという実装を行っています（①の箇所）。

ここで、画像を扱うために加えているのが「visibility:hidden」です。これは、「このオブジェクトを表示しない」というものです。なぜこの指定をしているかというと、初期設定により画像をWebブラウザの非表示領域に移動させて

いるのですが、画像の場合はロードに時間がかかったり、処理が遅いため最初に一瞬見えてしまう可能性があります。そこで、最初はレイヤーを見えないようにしておき、画像の移動が終わった段階で表示するという処理を行っています。これは、文字列に対して同様の記述をしても問題ありません。

次はスクリプトについてです。スクリプトは大きく分けて、初期設定と画像を移動させる部分の2つに分かれています。初期設定のinitPosition関数では、画像の移動位置や移動間隔などを変数で設定しています。さらに画像を初期位置に移動させています。初期位置は変数start_xが「-300」ピクセルとなっているので、Webブラウザの表示領域外の左側に隠れていることになります。

移動の方法は、すでにP.346で解説したとおりです。移動させた後、非表示になっているレイヤーを表示します。表示の切り替えにはvisibilityプロパティを用います。オブジェクトの指定がWebブラウザで異なるため、それぞれ次のようにしています。

●IE5以上、NN6

```
document.getElementById("subtitle").style.visibility =
⇒  'visible';
```

●NN4

```
document.layers["subtitle"].visibility = 'visible';
```

画像の移動が終了したら、moveImage関数を呼び出します。画像の移動は、文字列の移動とまったく同じですので、動作の仕組みは「11-5　文字を移動する」を参照してください。

T I P S

ここで紹介したサンプルでは、横方向に動かしましたが、縦方向に動かすこともできます。縦方向に動かしたい場合は、pixelLeftプロパティの代わりにpixelTopプロパティ、lefttopプロパティの代わりにtopプロパティを用います。

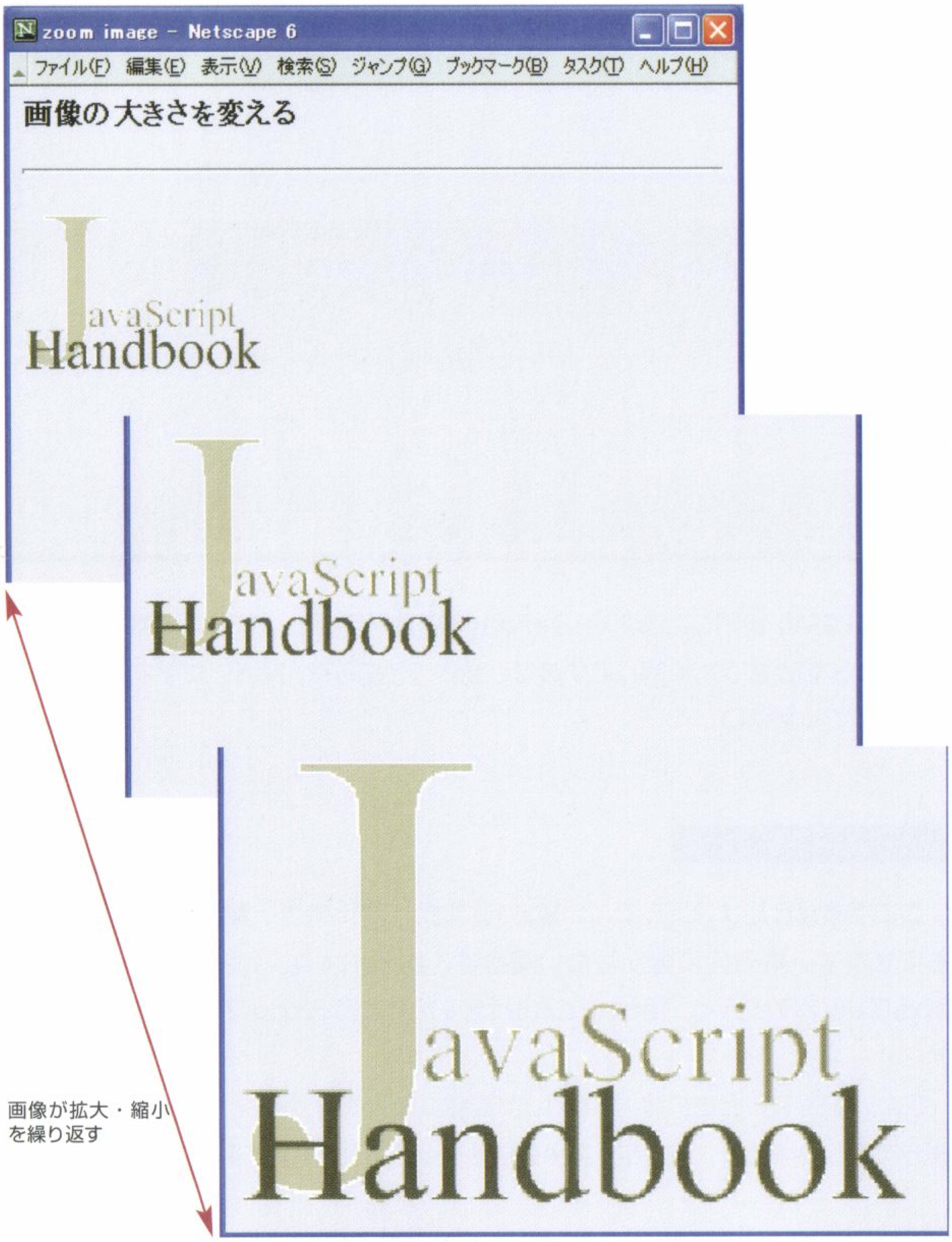
また、記述するときには、X座標とY座標にも気を付けるようにしてください。縦方向からの移動は、「11-5　文字を移動する」で扱っています。

画像の拡大・縮小

オブジェクト.setAttribute = (名前,値)

「11-6　文字の拡大・縮小」では、文字列を拡大・縮小させました。JavaScriptでは文字列だけでなく、画像も拡大・縮小することができます。画像でも、余りやりすぎるとセンスの悪いWebページになってしまいますが、ちょっとしたアクセントとして使うと、かなりよい感じを与えることができます。

DISPLAY



画像が拡大・縮小を繰り返す

SOURCE zoomimage.html

```
<html>
<title>zoom image</title>
<script language="javascript">
<!--
var i, dstep;
var iheight = 200;
var iwidth = 300;
var step = 0.01;
i=1.0;
dstep = step;

function zoomImage(){
   if( i>1.5 ){                                        ③
      dstep = -step;
   }
   if( i<0.5 ){
      dstep = step;
   }

   if(document.getElementById){                        ④
      document.getElementById("logoimage").            ⑤
         setAttribute('width', iwidth*i);
      document.getElementById("logoimage").
         setAttribute('height', iheight*i);
   }
   else if(document.layers){
      msg = '<img src="images/logo.gif" width="' +iwidth*i+
         '" height="' +iheight*i+ '">';
      document["subtitle"].document.open("text/html");
      document["subtitle"].document.write(msg);
      document["subtitle"].document.close();
   }
   i+=dstep;
   setTimeout("zoomImage()",100);
}
//-->
```

```
</script>
<body onload="zoomImage()" bgcolor="#e6e6fa">
<h3>画像の大きさを変える</h3>
<hr />
<div id="subtitle" style="position:absolute;top:80px"> ——②
   <img id="logoimage" src="images/logo.gif" /> ——①
</Div>
</body>
</html>
```

DETAILS

このサンプルでは、画像を拡大したり、縮小したりしています。スクリプトの基本的な構造は、「11-6　文字の拡大・縮小」と変わりません。サイズの指定の仕方が若干変わっている程度です。

このサンプルでもIE5以上とNN6、NN4ではそれぞれ処理が異なります。IE5以上とNN6ではイメージのプロパティを直接変更することにより、画像の拡大、縮小を実現します。画像を扱うために、①の箇所のように<img>要素のid属性で「logoimage」という名前を付けています。

NN4では直接イメージのプロパティが扱えないので、②の箇所のように<div>要素で作られたレイヤーの中に記述されています。そのため、レイヤーを操作すると画像もサイズが変更されます。このレイヤーの記述では、NN4でレイヤーとして認識させるために、id属性とstyle属性で「position:absolute」が必要になります。

次は画像のサイズ指定の方法です。文字列の場合は単純にフォントサイズを指定すればよかったのですが、画像には縦と横の長さがあります。さらに画像の縦横比を変えたくありません。そこで、このサンプルでは画像の縦と横の長さに同じ変数 i を掛けて、その変数iを変化させていくことにより画像の大きさを変えています。変数 i は、「0.5」から「1.5」まで動くようにしています。「0.5」より小さくなると、変数 i を変化させるために足していく変数dstepを正に、変数 i が「1.5」より大きくなると変数dstepを負にします(③の箇所)。

画像のサイズの変更は、IE5以上とNN6ではsetAttribute属性で指定します。IE5以上とNN6かどうかを、document.getElementByIdオブジェクトが動作するかどうかで確認します。動作する場合には、④の箇所のif文が真になるので、その次のブロックが実行されます。setAttribute属性は次のような書式になります。

```
オブジェクト.setAttribute(名前, 値)
```

これにより画像の幅と高さを直接指定しています。このサンプルで用いている画像は横300ピクセル、縦200ピクセルの画像です。その画像のサイズを⑤の箇所で変数iwidth、iheightに格納しています。そのため、画像のサイズの指定は以下のようになります。

●iwidth＊i
●iheight＊i

NN4の場合には、レイヤー内の文字列を変更することにより画像のサイズを変更します。レイヤーの内部に代入する文字列は<img>要素を用いて、この中でwidth属性とheight属性を指定します。IE5以上とNN6と同様に変数iを用いて表現された幅と高さをwidth属性、height属性に代入します。このようにして作成された文字列を格納した変数msgを、レイヤーの内部に対して書き込む処理を行うと画像が拡大または縮小されます。さらに、次の処理のために変数iに変数dstepを足し合わせます。

最後にこの一連の処理を繰り返して、連続的に画像のサイズを変更するために、setTimeoutメソッドを用いてタイマーを設定します。このサンプルでは100ミリ秒ごとに、zoomImage関数を呼び出すようにしています。

T I P S

NN4では画像の書き換え処理が遅いため、このサンプルを実行すると画像がちらついてしまうことがあります。そのため、あまりタイマーを短くすると見づらくなってしまいます。

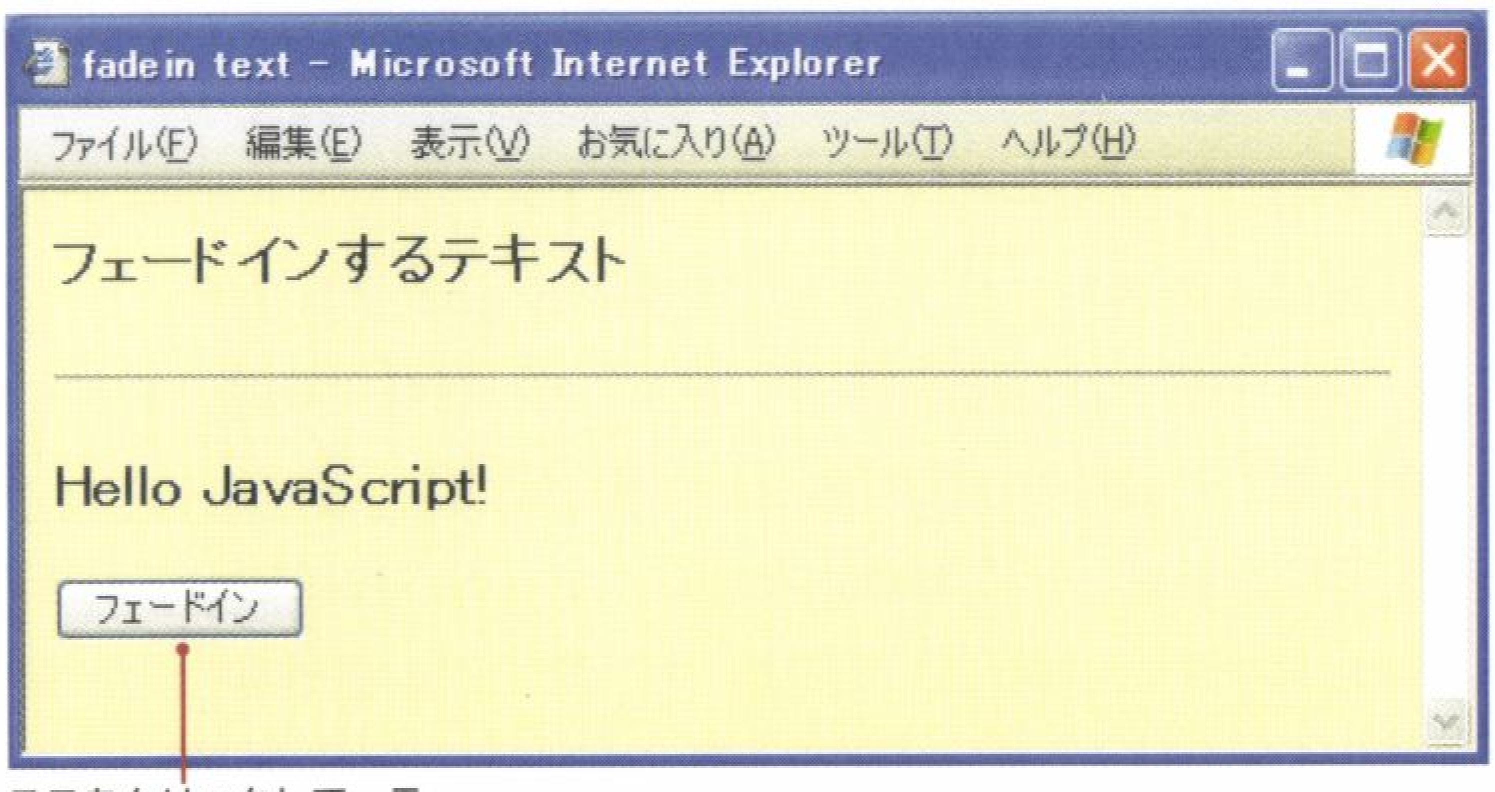
ローカルディスク内で動作確認した後に、インターネット上のWebページにスクリプトを配置し、実際に表示してみて、きちんと画像の拡大・縮小がスムーズにいくかどうかを確かめておくとよいでしょう。

フェードインするテキスト

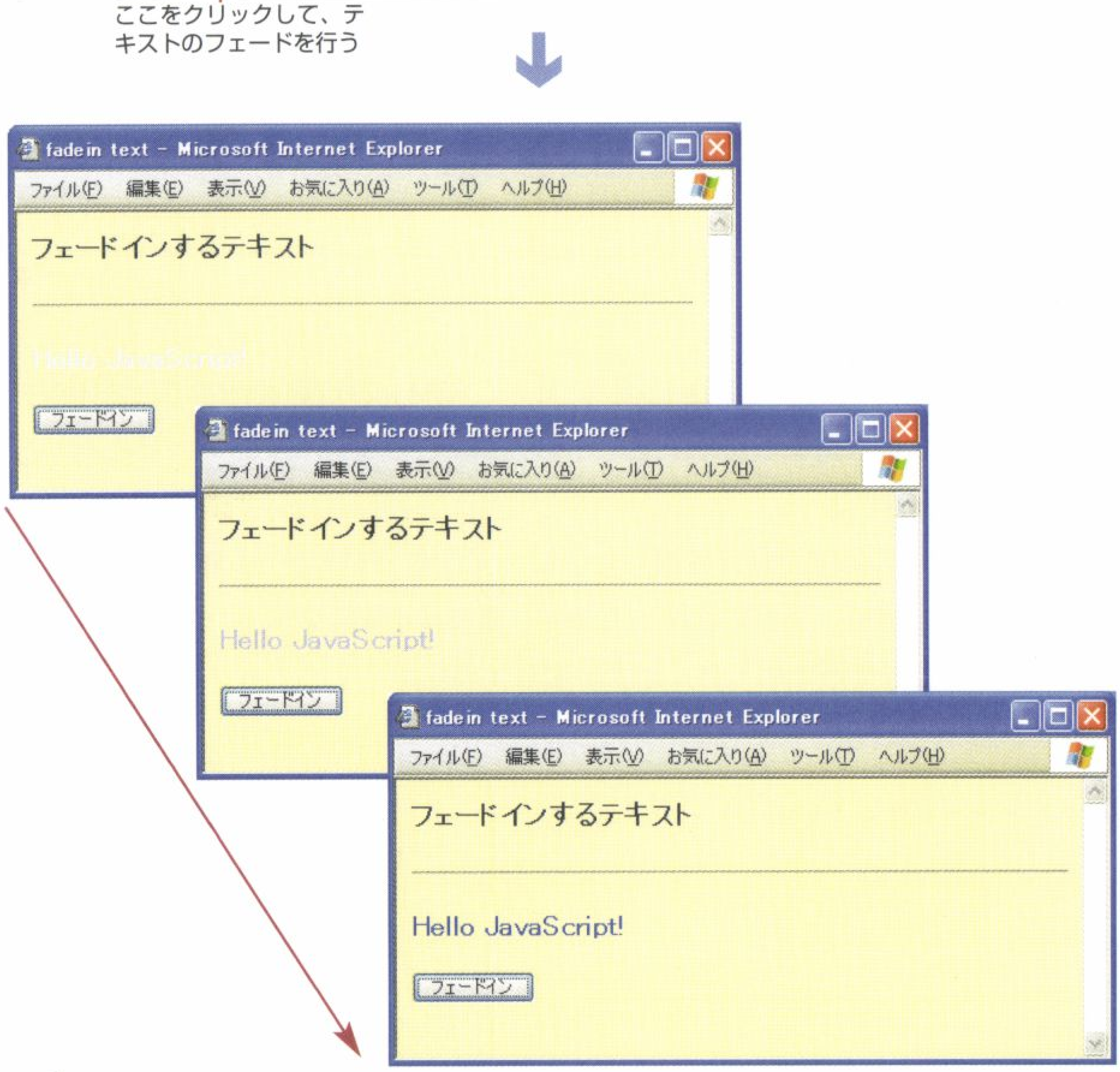
オブジェクトstyle.color= “rgb(“R” , “G” , “B”)”

このスクリプトでは、文字列をフェードインさせています。フェードインは目を引くので、ここぞという部分に使うと非常に効果を発揮します。効果的に使って、ユーザーの目を楽しませましょう。

D I S P L A Y



ここをクリックして、テキストのフェードを行う



SOURCE fadeintext.html

```
<html>
<title>fadein text</title>
<script language="JavaScript1.2">
<!--
var hex=255;
var step=8;

function fadeinText(){
   if(hex>0) {
      hex-=step;
      document.getElementById("fText").
         style.color="rgb("+hex+","+hex+","+255+")"; ——②
      setTimeout("fadeinText()",100); ——③
   }
   else{
      hex=255;
   }
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>フェードインするテキスト</h3>
<hr />
<div id="fText"><h3>Hello JavaScript!</h3></div>
<button onclick="fadeinText()">フェードイン</button>
</body>                                          ——①
</html>
```


DETAILS

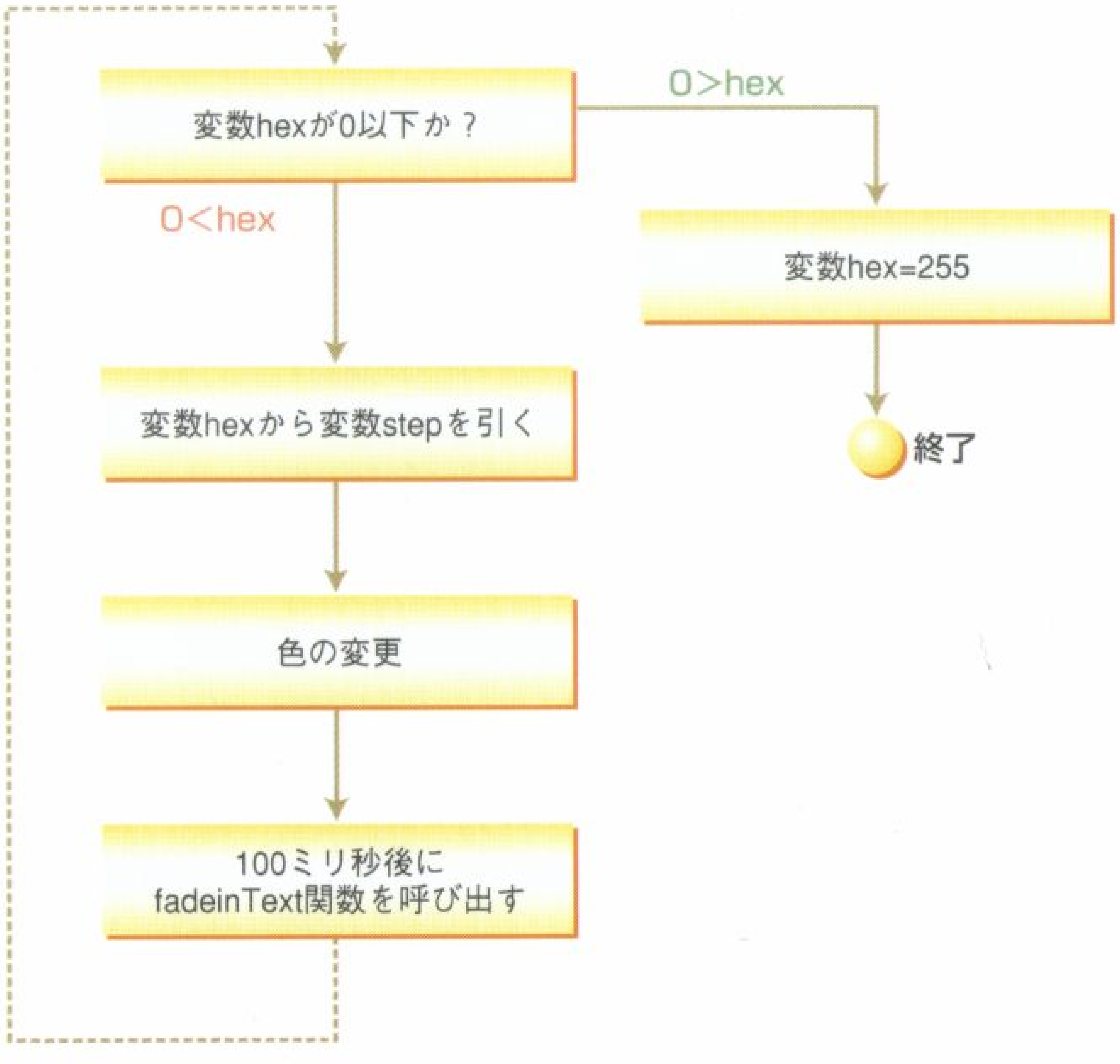
このスクリプトでは、「Hello JavaScript!」と書かれた文字列と、「フェードイン」ボタンが配置されています。動作としては［フェードイン］ボタンをクリックすると、文字がいったん消え、フェードインしてきます。

［フェードイン］ボタンを押すと、<button>要素に指定されたonclickイベントハンドラでfadeinText関数が呼び出されます（①の箇所）。

fadeinText関数では、色の指定に使う変数を変更しながら、colorプロパテ

ィを用いて色を実際に変更していく処理を行っています。

▼fadeinText関数の処理の流れ



まず、fadeinText関数では変数hexの値を調べています。変数hexが「0」以上の場合には色を変化させる処理を、「0」以下の場合には次に「フェードイン」ボタンが押されたときのために、変数hexを「255」に初期化します。

色を変更するにはcolorプロパティを用いています。②の箇所のように、colorプロパティに対して256進のRGBの値で色を指定しています。RGBの指定は以下のようになります。

```
rgb("赤の指定", "緑の指定","青の指定")
```

色は、rgb(255,255,255)が白、rgb(0,0,0)が黒になります。そして、rgb(255,0,0)が赤、rgb(0,255,0)が緑、rgb(0,0,255)が青です。ここでは、RGBのBだけが「255」の値で変わらず、RとGは、変数hexの値が255から段々と引かれていくことになるので、rgb(255,255,255)の白から少しずつ青に変化していきます。

色の設定をすると、③の箇所のようにsetTimeoutメソッドを用いて、100ミリ秒後に再度fadeinText関数を呼び出す指定をしています。これにより段々と色が変化していきます。

このサンプルでは、単純な色指定を行っていますが、色の指定を工夫することで、さまざまな変化を作り出すことができるはずです。

WIN IE 4↑ NN 4↑ MAC IE 4↑ NN 4↑

訪問者の名前を記憶する

document.cookie = クッキーに書き込む文字列

unescape（デコードする文字列）

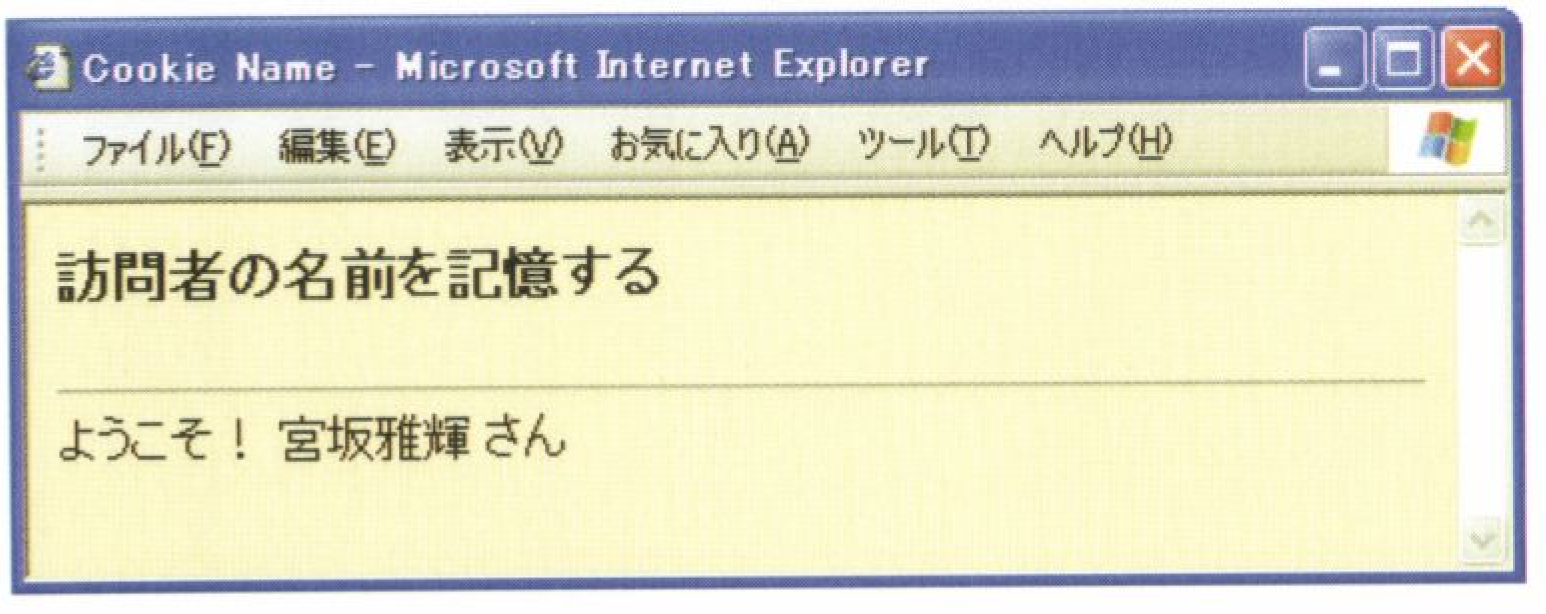
Cookie（クッキー）とは、Webブラウザがクライアント側のローカルファイルを参照する唯一の手段です。Cookieはテキストファイルで、文字列や数値などの情報を保存しています。これを使えば、さまざまな情報をWebページ側で表示することができます。ここでは、訪問したユーザー名を表示してみましょう。

DISPLAY

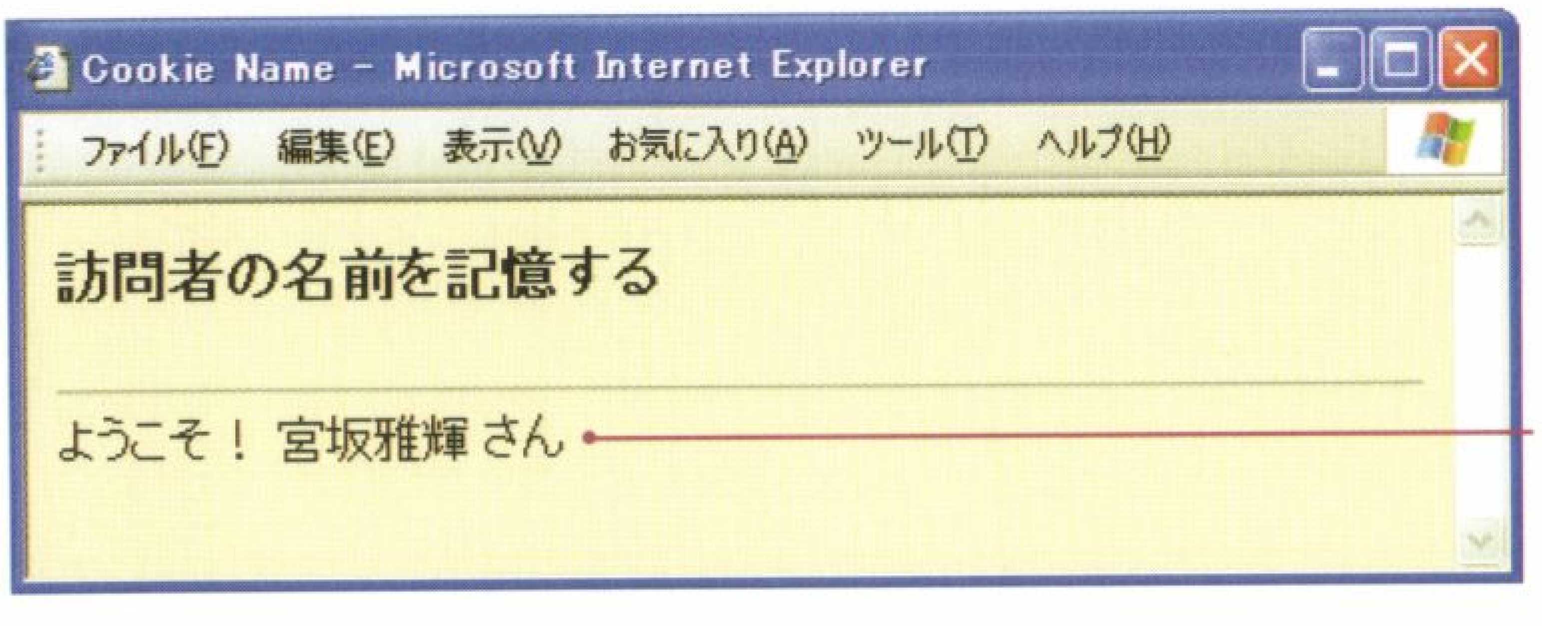
▼最初の訪問のとき



最初だけユーザー名を入力



▼2回目以降にアクセスした場合



以前の情報がクッキーから読み込まれて、ユーザー名が表示される

SOURCE cookie1.html

```
<html>
<title>Cookie Name</title>
<script language="javascript">
<!--
function getCookie(key){                                        ①
   tmp=document.cookie+";";
   tmp1=tmp.indexOf(key, 0);
   if(tmp1!=-1){
      tmp=tmp.substring(tmp1, tmp.length);
   start=tmp.indexOf("=", 0)+ 1;                                ⑦
      end=tmp.indexOf(";", start);
   return(unescape(tmp.substring(start, end)));
   }
   return("");                                                  ②
}
function inputName(){
   names=prompt("名前を入力してください。","Your Name");
   if(names==null){
      inputName();                                              ④
   }
   setCookie("Name", names);
}
function setCookie(key,val) {                                   ⑥
   tmp=key+"="+escape(val)+"; ";
   tmp+="expires=Fri, 31-Dec-2010 23:59:59 GMT; ";
   document.cookie=tmp;
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>訪問者の名前を記憶する</h3>
<hr />
<script language="javascript">
<!--
names=getCookie("Name");
if (names==""){                                                 ③
   inputName();
}
```

▼

COOKIE

```
document.write("ようこそ！ "+names+" さん");  ⑤
//-->
</script>
</body>
</html>
```

DETAILS

このサンプルでは、はじめてアクセスされたときに名前を入力してもらい、その情報をCookieに保存することによって、次回のアクセス時にその名前を画面に表示しています。

スクリプトが開始されると、まず①の部分のgetCookie関数が実行されます。

```
names = getCookie("Name");
```

この呼び出しのときに、パラメータとして「Name」を渡します。この関数によって返される値が、変数「names」に代入されます。後で詳しく説明しますが、getCookie関数では、渡されたパラメータを基に、Cookieファイルに保存されている情報から必要な部分を返します。ここでは、保存されているユーザー名が返され、変数namesに代入されます。はじめてアクセスした場合には空文字("")が返されるように設定してあります(②の箇所)。

③の部分のif文では、変数namesの値が空文字("")だった場合、つまりはじめてアクセスされた場合だけ処理を行います。はじめてのアクセスだった場合には、④の個所のinputName関数を呼び出しています。inputName関数では、プロンプトを表示して、名前を入力してもらいます。

入力された名前は変数namesに代入されます。namesが空だった場合は、もう一度inputName関数を呼び出します。namesに値が代入されたら、次にsetCookie関数を呼び出します。setCookie関数では、変数namesに代入されている値、つまり入力された名前をCookieファイルに登録しています。

最後に、変数namesに代入されている値を基に、ユーザーへのメッセージを画面に表示させています。⑤の部分が、次のようになっています。

```
document.write("ようこそ！ " + names + " さん");
```

ここからは、それぞれの関数について詳しく説明します。まずは、Cookieファイルから必要な情報を抜き出す、getCookie関数について説明します。getCookie関数では、はじめにCookieファイルの読み込みを行っています。

```
tmp = document.cookie + ";";
```

読み込まれたCookieファイルの内容は、変数「tmp」に代入されます。次にindexOfメソッドで、パラメータとして渡された文字列がCookieファイルにあるかどうかを検索し、あった場合はその先頭位置を変数tmp1に代入しています。なかった場合は「-1」が代入されます。

```
tmp1 = tmp.indexOf(key, 0);
```

keyには、パラメータとして渡された文字列が代入されていますので、その文字列を、変数tmpに代入されている文字列の0番目の位置から検索しています。

次のif文では、パラメータとして渡された文字列があったかどうかによって処理を分岐させています。はじめてアクセスされたときにはCookieファイルがありませんので、検索の結果として変数tmp1には「-1」が代入されます。その場合、if文の中の処理は行われず、次の部分が実行されます(②の箇所)。

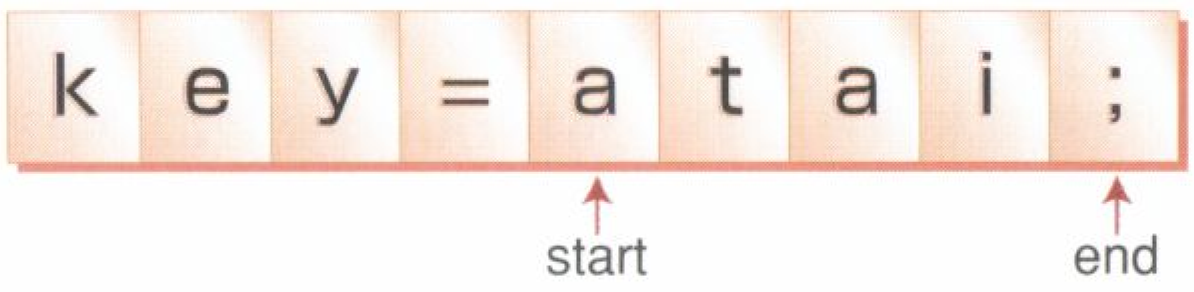
```
return("");
```

これにより、変数namesに空文字("")が代入されます。検索の結果、パラメータの文字列があった場合には、そのパラメータを基に、Cookieファイルに保存されている名前を取得します。

Cookie情報中に検索文字列があった場合には、⑥の部分の処理が行われます。

```
tmp = tmp.substring(tmp1, tmp.length);
start = tmp.indexOf("=", 0) + 1;
end = tmp.indexOf(";", start);
return(unescape(tmp.substring(start, end)));
```

まずsubstringメソッドを用いてキー以降の文字列を取り出し、変数tmpに格納します。次に、「キー=値;」となっている「=」の位置を取り出し、その後ろを変数startに格納します。また、「;」の位置を変数endに格納します。



この変数startと変数endを基に切り出し、unescapeメソッドで文字列のデコードを行って、その結果をgetCookie関数の呼び出し元に返します。文字列のデコードは、Cookieに格納される際に自動的にエンコードされてしまうためです。なおデコードに使うunescape関数は以下のような書式になります。

```
unescape(デコードする文字列)
```

Cookie情報に以前格納したデータがある場合には、getCookie関数を何度か用いることにより複数のクッキー情報を取り出します。

次に、getCookie関数によって返された値が空文字("")だった場合、つまりはじめてのアクセスだった場合に呼び出される、④の箇所のinputName関数について説明します。ここでは、プロンプトを表示し、名前を入力してもらいます。 キャンセル ボタンがクリックされた場合には、再びinputName関数を呼び出して、入力されるまでプロンプトの表示を繰り返します。入力された値は、変数namesに代入されます。その後で、「Name」という文字列と変数namesをパラメータとして、setCookie関数を呼び出しています。

最後に、⑦の部分のsetCookie関数について説明します。この部分では、Cookieファイルへの書き込みを行っています。パラメータとして渡された値によって、変数「key」にはNameが、変数「val」には「設定した名前」が代入されます。処理は、次の3行で行われています。

```
tmp = key + "=" + escape(val) + "; ";
tmp += "expires=Fri, 31-Dec-2030 23:59:59 GMT; ";
document.cookie = tmp;
```

1行目で、変数tmpには、「Name=設定した名前」が代入されます。2行目では、その文字列に、Cookieの有効期限を表す文字列が連結されます。期限を指定しなかった場合、Webブラウザを終了するとクッキーが消去されてしまいますので注意してください。最後にdocumentオブジェクトのcookieプロパティを使って、変数tmpの値をクッキーに書き込んでいます。

12-2 WIN 4↑ N 4↑ MAC 4↑ N 4↑

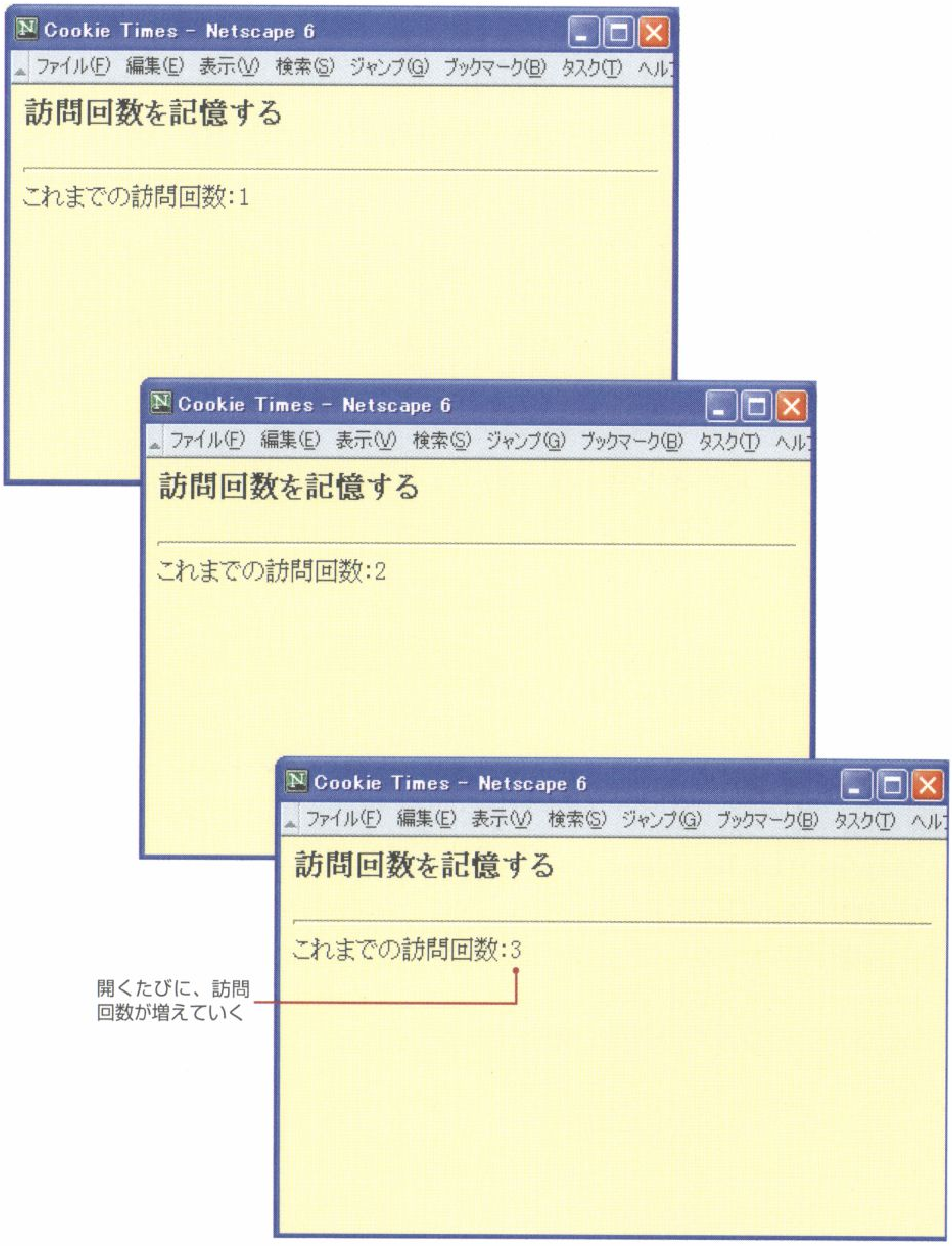
ユーザーが何回アクセスしたかを記憶する

document.cookie = クッキーに書き込む文字列

unescape（デコードする文字列）

Cookieには、ユーザーがそのWebページを何回訪れているかといった情報も保存されています。「あなたは……回目の訪問です」といった表示をしてみましょう。

DISPLAY



SOURCE

cookie2.html

```
<html>
<title>Cookie Times</title>
<script language="javascript">
<!--
var times;
function getCookie(key){
   tmp=document.cookie+";";
   tmp1=tmp.indexOf(key, 0);
   if(tmp1!=-1){
      tmp=tmp.substring(tmp1,tmp.length);
         start=tmp.indexOf("=",0)+1;
      end=tmp.indexOf(";",start);
         return(unescape(tmp.substring(start,end)));
   }
   return("");
}

function setCookie(key,val) {
   tmp=key+"="+escape(val)+";";
   tmp+="expires=Fri, 31-Dec-2010 23:59:59; ";
   document.cookie=tmp;
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>訪問回数を記憶する</h3>
<hr />
<script language="javascript">
<!--
times=getCookie("Times") ―――――――――― ①
if (times=="") { ―――――――――――――― ②
    times=1;
}
document.write("これまでの訪問回数："+times);
times++; ――――――――――――――――――― ③
setCookie("Times",times);
//-->
```

```
</script>
</body>
</html>
```

DETAILS

getCookie関数とsetCookie関数の2つは、先ほどの「12-1　訪問者の名前を記憶する」の名前を表示するサンプルとまったく同じです。ここでは、先ほどのサンプルと違う部分だけを説明します。

①の箇所でgetCookie関数を呼び出しています。与えるパラメータはCookieのキーとなる文字列「Times」です。このパラメータによってCookie内の他の情報と区別され、必要な情報だけが取得できます。ここでは、今までのアクセス回数が返され、変数「times」に代入されます。

はじめてのアクセスだった場合、getCookie関数によって空文字("")が返されます。その場合には、②の箇所のif文の処理が実行され、変数timesには「1」が代入されます。

次に「これまでの訪問回数:」という文字列に、timesに、代入されている値を結合したものを画面に表示させています。

その後、③の部分で変数timesの値を「1」増やしてから、setCookie関数を呼び出します。パラメータは文字列「Times」と変数timesの値です。Cookieファイルには「1」増えた後の変数timesの値が書き込まれるので、次にアクセスしたときには1つ値が増えているのです。

このように、クッキーを使って、ユーザーが訪れている回数をWebページに表示することができます。回数に応じて、「はじめまして」や「いつもありがとう」など、メッセージを変えて表示してみるのも面白いでしょう。

12-3 WIN MAC

名前と訪問回数を記憶する

クッキーから読み出した文字列を代入する変数==document.cookie

unescape（デコードする文字列）

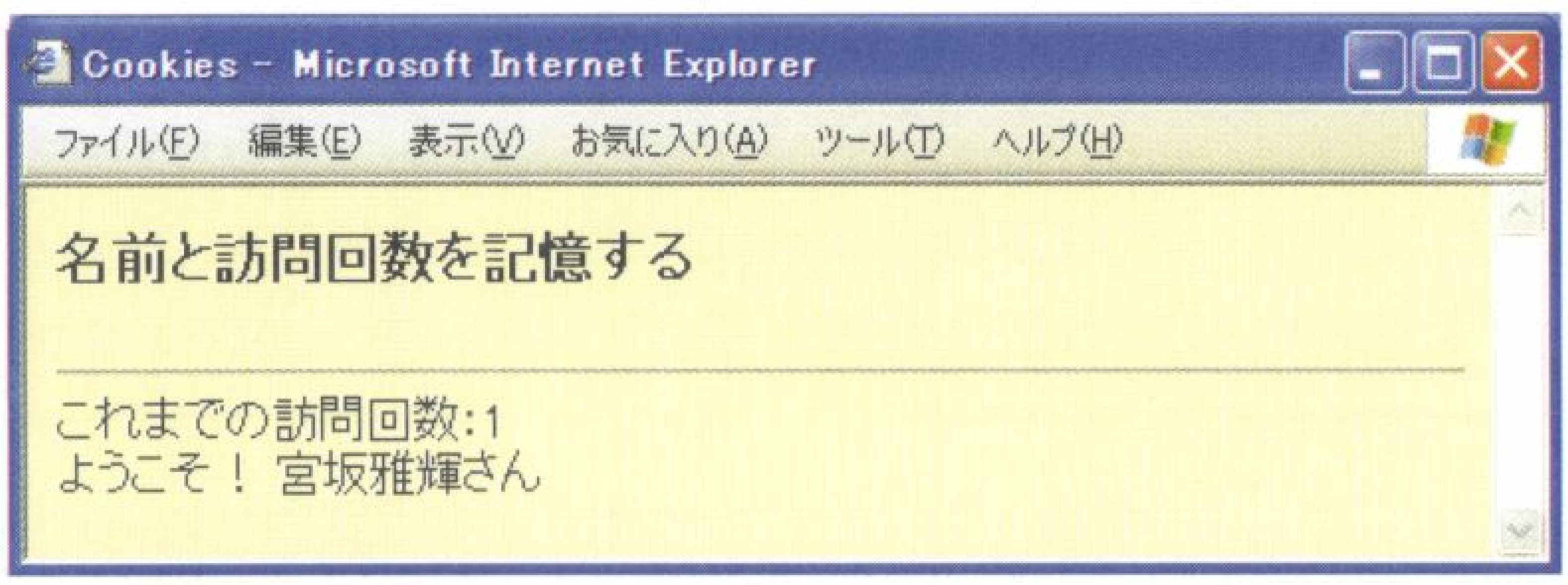
このサンプルは、これまでに説明した、Cookieに名前を保存するスクリプトと、Cookieにアクセス回数を保存するスクリプトを組み合わせたものです。複数の情報を保存させておきたい場合に参考にしてください。

DISPLAY

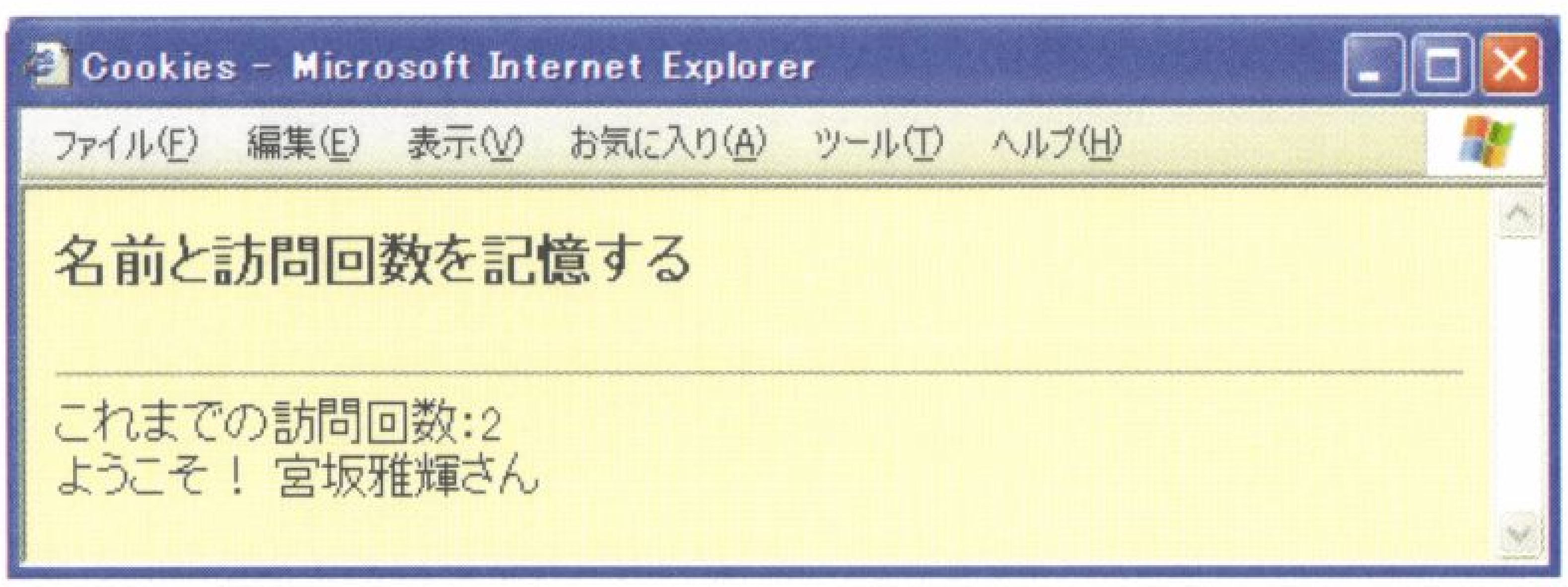
▼最初の訪問のとき



最初だけ、ユーザー名を入力する



▼2回目以降の訪問のとき



SOURCE

cookie3.html

```
<html>
<title>Cookies</title>
<script language="javascript">
<!--
function getCookie(key)
    tmp=document.cookie+";";
    tmp1=tmp.indexOf(key,0); ——②
    if(tmp1 !=-1){{
        tmp=tmp.substring(tmp1,tmp.length);
            start=tmp.indexOf("=",0) + 1;
        end = tmp.indexOf(";",start);
            return(unescape(tmp.substring(start,end)));
    }
    return("");
}
function setCookie(key,val){
    tmp=key+"="+escape(val)+"; ";
    tmp+="expires=Fri,1-Jan-2010 23:59:59; ";
    document.cookie=tmp;
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>名前と訪問回数を記憶する</h3>
<hr />
<script language="javascript">
times=getCookie("Times");
if (times=="") {
    times=1;
}
names=getCookie("Name");
if (names=="") {
    inputName();
}
document.write("これまでの訪問回数："+times+"<BR>"); ——①
document.write("ようこそ！ "+names+"さん");
times++;
setCookie("Times",times);
```

▼

```
setCookie("Name",names);
function inputName(){
    names=prompt("名前を入力してください。","Your Name");
   if(names==null){
      inputName();
   }
}
</script>
</body>
</html>
```

DETAILS

このサンプルを読み込むと、今までにアクセスした回数と、初回アクセス時に入力した名前が、ページ上に表示されます。

①の箇所でページ上に書き込む際に改行するための、
タグを加えていること以外は、「12-1　訪問者の名前を記憶する」と「12-2　ユーザーが何回アクセスしたかを記憶する」の2つのサンプルをコピーして貼り付けただけで、どこにも手が加えられていません。これは、Cookieを読み込んだり書き込んだりするために作成したgetCookie関数とsetCookie関数が、パラメータによって必要な部分だけを参照できるように作成されているからです。

getCookie関数では、②の部分で、パラメータとして与えられた「Times」または「Name」の先頭位置を調べ、その位置から処理を行っています。引数として与えられた文字列は、変数keyに格納されています。

```
tmp1 = tmp.indexOf(key, 0);
```

これによって、Nameの値を参照したい場合には、Timesに保存されている部分を参照することなく、Name以下だけが参照され、正しい結果を返すことができます。

このように複数の値をCookieに保存して用いることができます。

12-4 WIN 4↑ N 4↑ MAC 4↑ N 4↑

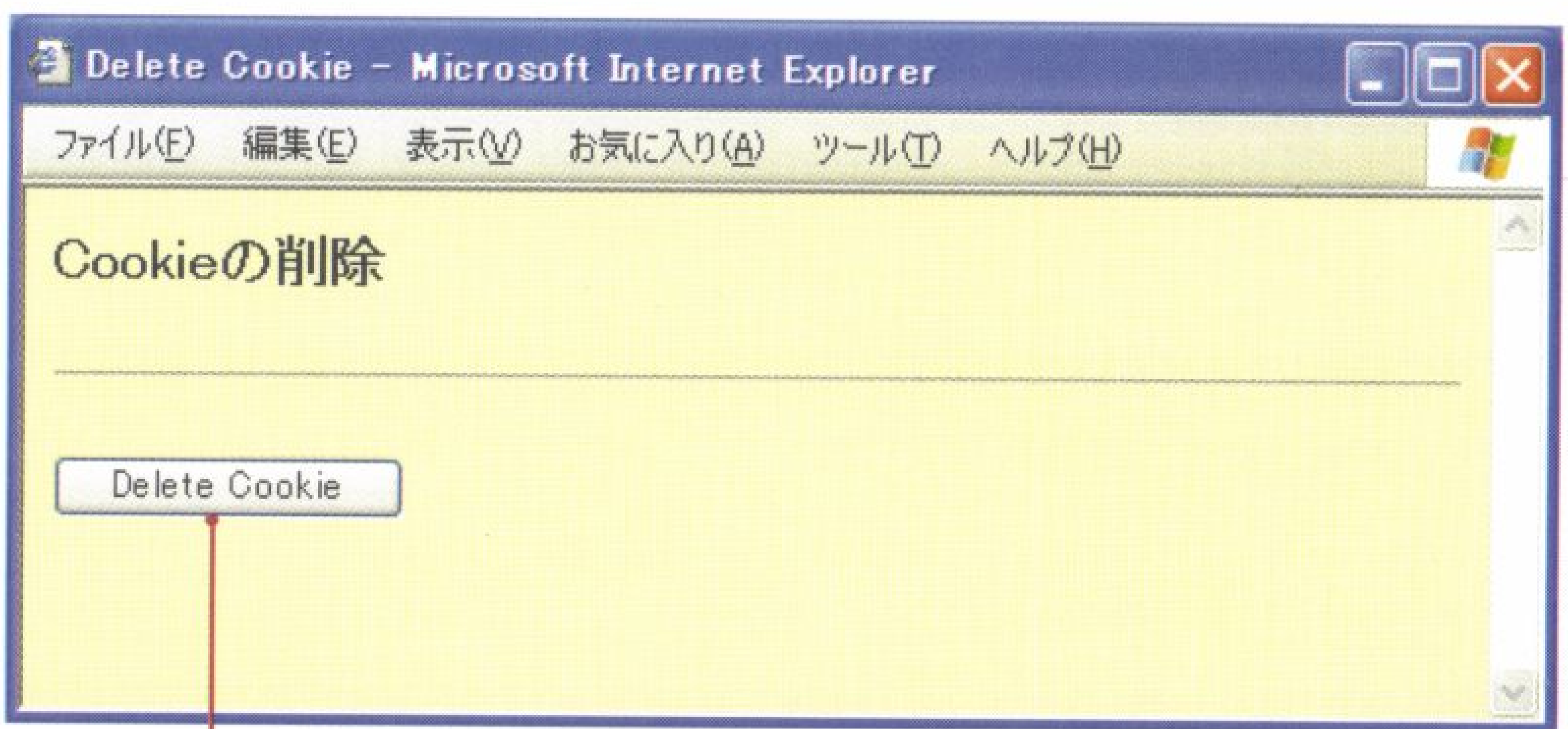
Cookieを削除する

`Dateオブジェクト名.setYear (Dateオブジェクト.getYear()-1)`

このスクリプトはCookieで設定したものを削除するものです。何らかの理由で、Webブラウザに送ったCookieを削除する必要があるかもしれません。また、データ形式の修正が入った場合などには、Cookie消してから、再度他の値を書き込むといった処理も考えられます。ここではCookieを削除するスクリプトを紹介します。

D I S P L A Y

▼削除するCookieがあった場合



ここをクリックして、
Cookieを削除する



▼削除するクッキーがなかった場合



SOURCE deletecookie.html

```
<html>
<title>Delete Cookie</title>
<script language="javascript">
<!--
function deletelCookie(){
    if(document.cookie != "" ){                                   ①
        tmpCookie = document.cookie.split(";");
        expireDate = new Date();                                  ②
        expireDate.setYear(expireDate.getYear() -1);
        for(i=0; i<tmpCookie.length; i++){                        ④
            cName = tmpCookie[i].split("=")[0];
            document.cookie = cName + "=;                         ③
            expireDate=" +expireDate.toGMTString();
        }
        alert("Cookieを"+tmpCookie.length+"個削除しました。");
    }
    else{
        alert("削除するCookieはありません。");
    }
}
//-->
</script>
<body bgcolor="#fff8dc">
```

```
<h3>Cookieの削除</h3>
<hr />
<form>
    <input type="button" value="Delete Cookie"
⇒       onclick="deletelCookie()" />
</form>
</body>
</html>
```

DETAILS

このスクリプトではCookieを削除します。Cookieの削除は非常に簡単です。Cookieの有効期限を現在の日付よりも過去にすれば自動的にCookieは削除されます。

まず最初に、①の箇所で行われているようにCookieが設定されているかどうかを調べます。ここでもしCookieが空("")だったならば、このif文は真にならないので処理は行われません。Cookieがセットされているとしたら、splitメソッドを用いて「;」で切り離した値を変数「tmpCookie」に代入します。

次にCookieの有効期限を割り当てるためのDateオブジェクト「expireDate」を用意します。ここではDate関数を使って現在の日付を取得した後、expireDateの年から「1」を引いた値を、再度expireDateにセットしています(②の個所)。

ここまでの処理により、expireDateは現在の日付より1年前の日付がセットされています。

最後に、③の箇所のように"document.cookie"に再度CookieをexpireDateの日付を使って設定し直すことにより、Cookieを削除します。

なお、このサンプルでは再度日付に変換する際に、toGMTStringメソッドを用いています。これはグリニッジ標準時を文字列で返すメソッドです。同様の世界協定時を返すtoUTCStringメソッドを用いてもかまいません。

●グリニッジ標準時を返すメソッド

```
Dateオブジェクト.toGMTString();
```

●世界協定時を返すメソッド

```
Dateオブジェクト.toUTCString();
```

また、④の記述によりこのサーバーが送り込んだすべてのCookieを削除することができます。この処理はfor文によりCookieの数だけ処理を繰り返すものです。

削除するCookieがない場合は、「削除するCookieはありません」というアラートが表示されます。

なお、このCookieを削除するサンプルは、削除したいCookieを設定したWebページが配置してあるフォルダと同じフォルダになければなりません。

COLUMN

P3P

インターネット上でさまざまなサービスが行われるようになったことで、話題になっているのが、個人情報の問題、そうプライバシーに関わる部分です。しかし、さまざまな経済活動がインターネット上で行われると、当然個人情報が必要な場面に遭遇します。このような状況の中でW3CはP3P（The Platform for Privacy Preferences）という規格を検討しています（2001年11月現在）。

この検討中の規格をIE6が部分的にサポートしています。それはクッキーの扱い方の部分です。IE6では、ディフォルト状態でサードドメインのクッキーをブロックするようになりました。要するにWebページ内に埋め込まれた外部サイトのiframeやimageの要素からのクッキーは受け付けないということです。

さらに、P3Pに対応していない場合には、ステータスバーに警告が出るようにしています。このような実装は、Cookieを使ってインターネット上での個人のトラッキングなどが行われていると考えているためです。これによって困るのは広告などを提供している広告配信会社になります。

ではどのようにすればP3Pに対応したといえるのでしょうか？　まずは当然ですが、個人情報取り扱い規定（Privacy Policy Statement）を作成します。その上で、HTTPヘッダーとWebサイト上の個人情報取り扱い規定に記載されている定義を設定します。

なお、サイトをどのようにP3P対応にするかについての詳細な説明は、次のURLを参照してください。

URL▶ http://www.w3.org/P3P/details.html

URL▶ http://www.w3.org/TR/p3pdeployment

Section 13 OTHERS

13-1 WIN 4↑ 4↑ MAC 4↑ 4↑

外部ファイルからリストを作成する

このサンプルは外部JavaScriptファイルを使って、特定のWebブラウザ用にURLのリストを作るというものです。HTMLファイルのほかに「＊.js」ファイルを用意して、それを取り込みます。

D I S P L A Y



Webブラウザによって、表示されるリンクが変更される

SOURCE

exdblist.html

```
<html>
<title>External List by JavaScript</title>
<style type="text/css">
   td.categorycell { background: #dddddd; }
   a:link { color: #000080;
      text-decoration: none }
   a:visited { color: #800000;
      text-decoration: none }
   a:hover {   color: #ff0000;
      text-decoration: underline}
</style>
<script language="javascript" src="url.js"></script> ―①
<script language="javascript">
<!--
var cols = 2; ―④
var rows = 10;
var amount = cols * rows;
var ua;

function makeList(){
   var uagent = navigator.userAgent;
   if( uagent.indexOf('MSIE') >= 0 ){        ┐
      ua = "IE";                             │
   }                                         │
   else if( uagent.indexOf('Mozilla/') >= 0 ){│
      ua = "NN";                             ├②
   }                                         │
   else{                                     │
      ua = "Other";                          │
   }                                         ┘

   document.write("<h4>これは" + ua + "用のリンクです</h4>");
   document.write("<table border='0'>");
   for( i = 0 ; i < amount ; i+=cols ){ ―⑤
      if( exDB[i] == ua || exDB[i] == "BOTH" ){ ―③
         document.write("<tr><td class='categorycell'
            width='60'>" + exDB[i] + "</td>");
```

```
        document.write("<td><a href=" + exDB[i+1] +">"
        + exDB[i+1] + "</a></td></tr>");
      }
    }
    document.write("</table>");
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>外部ファイルからリストを作成する</h3>
<hr />
<script language="javascript">
<!--
    makeList();
//-->
</script>
</body>
</html>
```

url.js

```
var exDB = new Array();
exDB[0] = "IE";
exDB[1] = "http://www.microsoft.com/";
exDB[2] = "NN";
exDB[3] = "http://home.netscape.com/";
exDB[4] = "BOTH";
exDB[5] = "http://www.softbank.co.jp/";
exDB[6] = "IE";
exDB[7] = "http://www.microsoft.com/japan";
exDB[8] = "BOTH";
exDB[9] = "http://www.yahoo.co.jp/";
exDB[10] = "BOTH";
exDB[11] = "http://www.neko-no-te.net/books/";
exDB[12] = "NN";
exDB[13] = "http://www.netscape.com/ja/";
exDB[14] = "IE";
exDB[15] = "http://www.msn.co.jp/";
exDB[16] = "BOTH";
```

```
exDB[17] = "http://www.zdnet.co.jp/"

exDB[18] = "BOTH";
exDB[19] = "http://www.geocities.com/";
```

DETAILS

このサンプルはWebブラウザの判定を行い、IE、NNごとに異なるURLのリストを出力するスクリプトです。Webブラウザごとに出力するものを変えるというのはよくある話です。このとき、通常ならば1つのHTMLファイル中に両方のコードを書いて、それを判定させて出力することが多いのですが、結構メンテナンスが面倒だったりします。そこで、リストに相当する部分だけ外部に切り出して、メンテナンス性をよくします。また、このようにしておくと、外部ファイルの部分を切り替えることで、すぐに更新作業ができるようになります。

では、最初に外部JavaScriptファイル「url.js」の説明をします。外部JavaScriptファイルは当然ながらJavaScriptのコードでなくてはなりませんので、データを扱うのに便利なCSV形式（カンマ区切り）やタブ区切りは扱うことができません。しかし、変数を多数用意するのも面倒なので、配列要素にデータを代入するという方法をとります。

そのため、まず配列要素を宣言します。

```
var exDB = new Array();
```

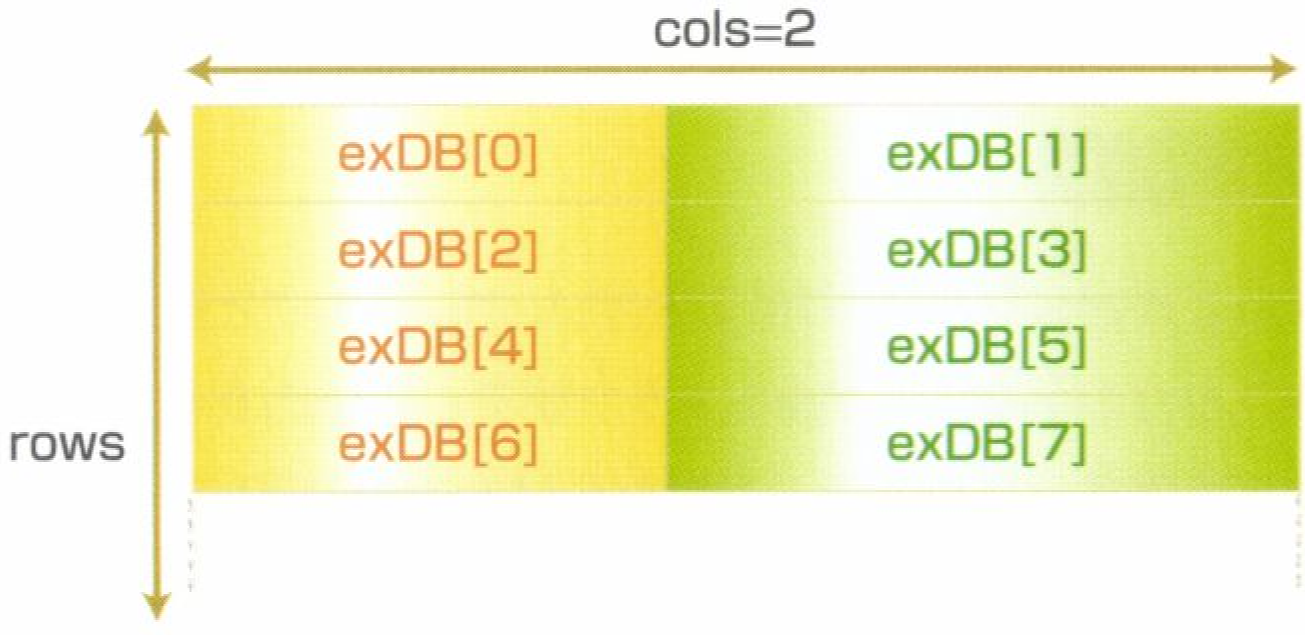
その後、この配列にデータを入れていきます。

```
exDB[添え字] = データ;
```

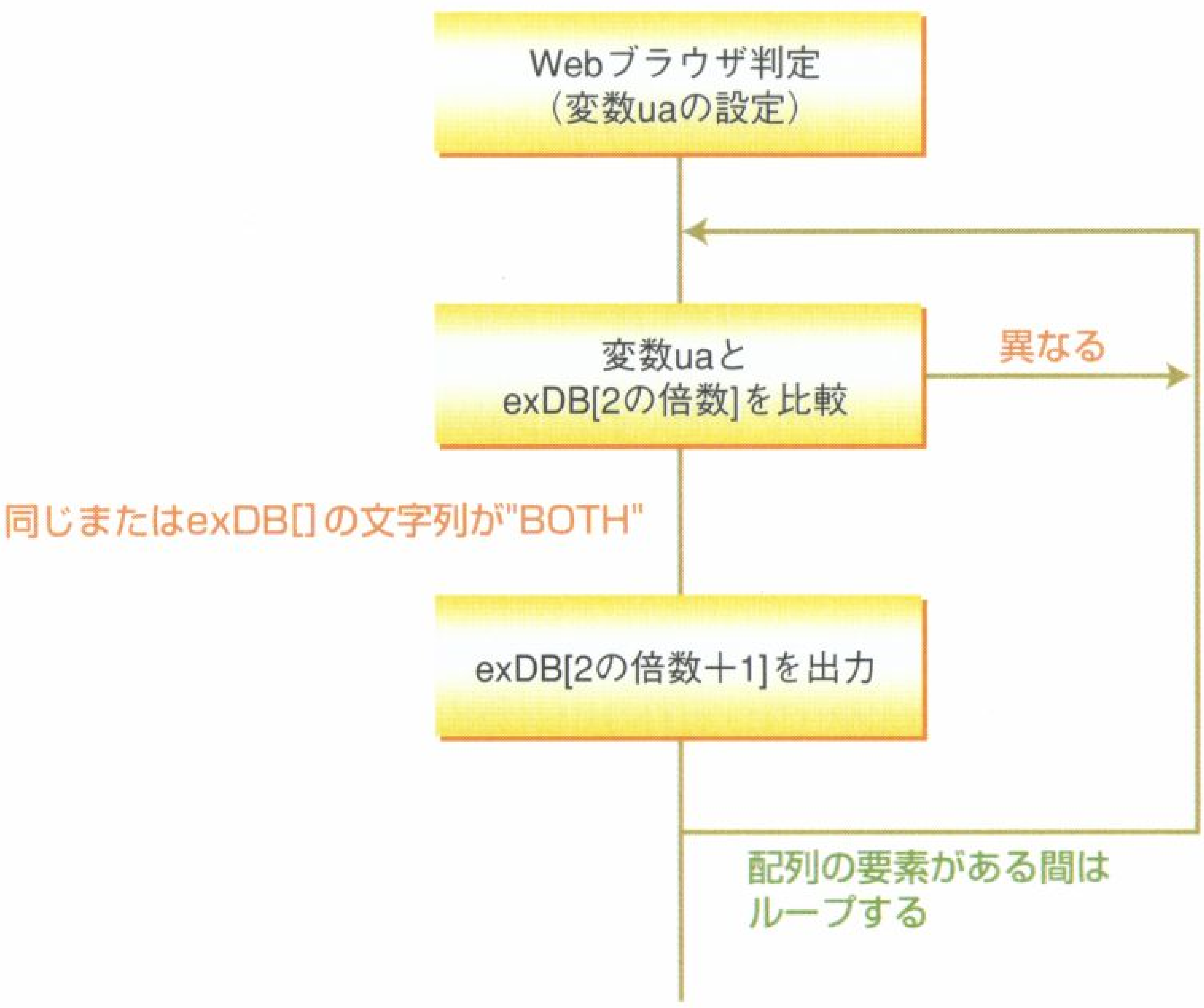
サンプルではどのWebブラウザ用かという情報と、URLの情報の2つを1組のデータとして扱うようにしています。そのため、横方向2のテーブルと考えることもできます。外部JavaScriptファイルはNN3以降、IE4以降でしか動作しません。またファイル名は拡張子が「.js」でなくてはなりません。これは必須なので、Macintoshなどの制作環境の方は忘れないようにしましょう。

次に本体のHTMLファイルについて見ていきましょう。最初に<script>タグのsrc属性を用いて、①のように外部ファイルに対してリンクを行っています。実際にこの部分にファイルのスクリプトが読み込まれると考えると理解しやすいと思います。

次に外部ファイルの内容をテーブルとして見たときの縦(rows)と横(cols)の数を設定します。なお外部JavaScriptファイルをどのようなテーブルとして見るかは以下の図のとおりです。また要素の全体の数をcolsとrowsの積で求めます。



さらに、ユーザーのWebブラウザを判定して、Webブラウザごとリストを生成するためのmakeList関数を用意します。この関数は、②でWebブラウザの判定を行い、その結果を変数「ua」に格納します。次にuaと外部JavaScriptファイル内の配列で添え字が2の倍数の要素内に入っている文字を、③のように比較します。なぜ2の倍数なのかというと、④で変数colsに2を代入しているからです。このとき配列要素内の文字列とuaが同じ場合、または配列要素内の文字列が「BOTH」だった場合には表示するようにしています。処理の流れを図示すると次のようになります。



なお、ループの処理ですが、配列の要素の数が変数「amount」に格納されているので、これを終了条件とします。また配列はカラムの数ごとにWebブラウザに関しての情報が入っているので、継続条件は変数colsを足します。すなわちカラム数ごとに参照していくのでfor文の記述は⑤のようになります。

このように、外部JavaScriptファイルを使うと比較的すっきりとしたスクリプトで上記のような機能を実装することができます。データの部分と処理の部分を分離して記述するというのはプログラミング上有効であるといえます。また外部JavaScriptファイルは複数用いることができるので、処理の部分を分割するといったことも可能となります。こうしておくとメンテナンス性がよいだけでなく、スクリプトを使い回すこともできます。

ただし、この場合には本体のHTMLファイルの他に外部JavaScriptファイルをダウンロードしなくてはなりません。そのため、Webサーバーにかかる負荷が大きくなったりします。さらに、外部JavaScriptファイルで扱うデータがあまり多いと不安定になることがあります。これは、いくつといった明確な制限があるわけではなく、OSや搭載メモリ、割り当てメモリ、ハードディスク容量、Webブラウザ、Webブラウザのバージョンなどによって異なります。しかし、少なくとも1000行程度までは安定して動作しているので、簡単なものならあまり問題とならないでしょう。

COLUMN

JavaScriptの応用範囲

JavaScriptはWebページ上のさまざまな要素を操作することができます。しかし、それだけではなくプラグインやJavaなども操作することができます。

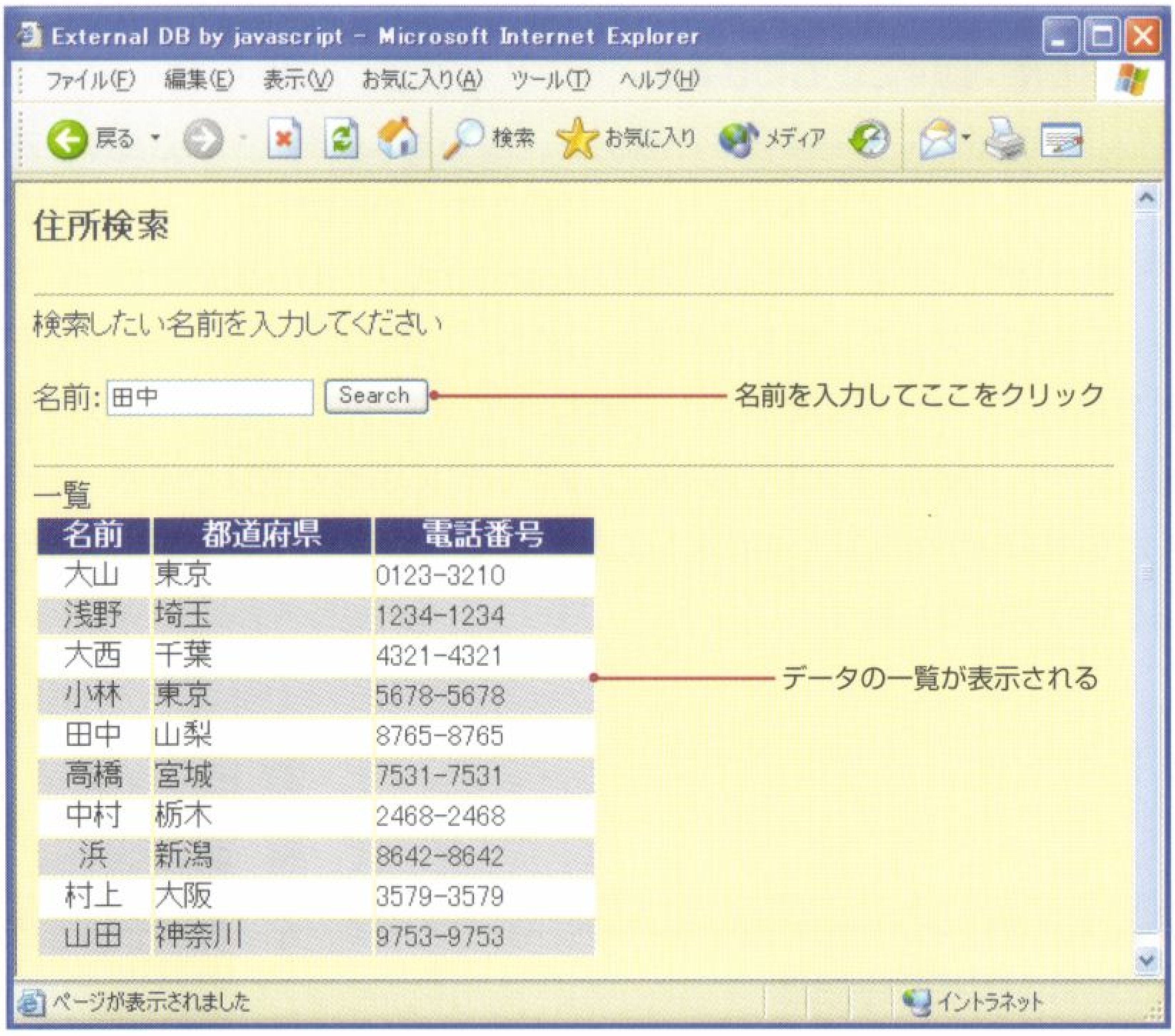
もっとも代表的なのはMacromedia社のFlashでしょう。ムービーの再生、停止、フレームの指定などをJavaScriptから行うことができます。これにより、Flashのムービーと連携できるとともに、Flashだけでは難しかった動作も実現できることになります。これはMacromedia社のShockwaveについても同様です。これらの機能は「Live Connect」と呼ばれています。

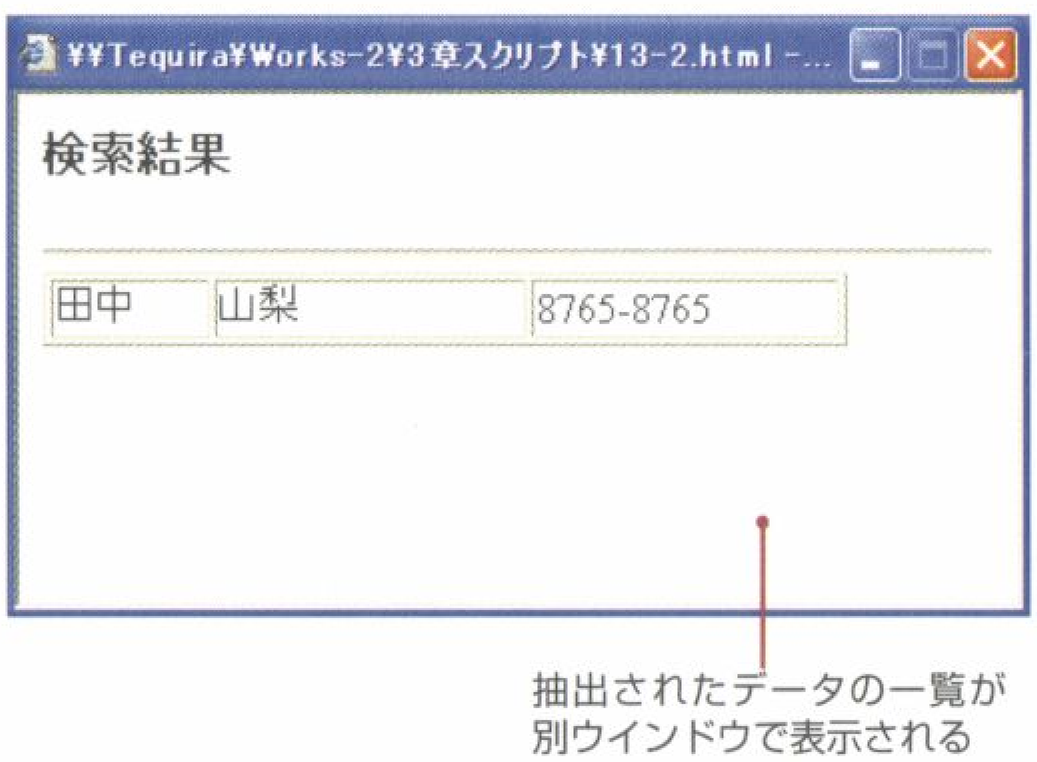
また、Javaアプレットで公開されているメソッドを操作することもできます。Javaアプレットの作り方次第ですが、HTMLで書かれたフォームから、Javaアプレットを操作することが可能となります。

住所録を検索する

外部JavaScriptファイルを使った簡易データベースです。Webページにたくさんのリストやデータを表示し、外部JavaScriptファイルを用いた簡易検索システムを実装することができます。

DISPLAY





SOURCE searchaddress.html

```
<html>
<title>External DB by javascript</title>
<style type="text/css">
<!--
th {  background: #000080;
      color: #ffffff }
.light-center { background: #ffffff;
      text-align: center }
.light { background: #ffffff }
.dark-center { background: #dddddd;
      text-align: center }
.dark { background: #dddddd }
-->
</style>
<script language="javascript" src="./data.js"></script>
<script language="javascript">
<!--

var i;
var cols = 3;
var rows = 10;
var amount = cols * rows;

function dataSearch(ichar){
   if( ichar == "" ){ ────────────────────────── ②
      alert("検索する文字列を入力してください");
      document.iform.ichar.focus();
      return false;
   }
   var flag = false; ────────────────────────── ③
   var resultWindow = window.open("","Result","toolbar=no,
⇒     location=no, status=no, menubar=no, width=400,
⇒     height=200");
   resultWindow.document.write("<h3>検索結果</h3>");
   resultWindow.document.write("<hr /><table border='1'>");
   for( i = 0 ; i < amount ; i+=cols ){
      if( exDB[i] == ichar ){ ──────────────────── ④
```

▼

OTHERS

```
            resultWindow.document.write("<tr><td width=60>" +
                exDB[i] + "</td>")

            resultWindow.document.write("<td width=120>" +
                exDB[i+1] + "</td>");
            resultWindow.document.write("<td width=120>" +
                exDB[i+2] + "</td></tr>");
            flag = true; ――――――――――――――――――――⑤
        }
    }
    if( flag == false ){ ――――――――――――――――――――⑥
        resultWindow.document.write("<tr><td>該当する文字列はありません
            </td></tr>");
    }
    resultWindow.document.write("</table>");
    resultWindow.document.close(); ――――――――――――――――⑦
}

function printData(){
    document.write("<table border='0'>");
    document.write("<tr><th>名前</th><th>都道府県</th><th>電話番号
        </th></tr>");
    for( i=0 ; i<amount ; i+=cols ){
        if( i%2 == 0 ){

            document.write("<tr><td width='60' class='light-
                center'>" + exDB[i] + "</th>");
            document.write("<td width='120' class='light'>" +
                exDB[i+1] + "</td>");
            document.write("<td width='120' class='light'>" +
                exDB[i+2] + "</td></tr>");
        }
        else{
            document.write("<tr><td width='60' class='dark-
                center'>" + exDB[i] + "</th>");
            document.write("<td width='120' class='dark'>" +
                exDB[i+1] + "</td>");
            document.write("<td width='120' class='dark'>" +
                exDB[i+2] + "</td></tr>");
        }
```

```
    }
    document.write("</table>");
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>住所検索</h3>
<hr />
検索したい名前を入力してください<br />
<form name="iform">                                         ┐
    名前：<input type="text" name="ichar" />                  │
    <input type="button" value="Search"                       ├─①
⇒       onClick="dataSearch(document.iform.ichar.value)" />   │
</form>                                                       ┘
<hr />
一覧<br />
<script language="javascript">
<!--
    printData();
//-->
</script>
</body>
</html>
```

data.jp

```
var exDB = new Array();
exDB[0] = "大山";
exDB[1] = "東京";
exDB[2] = "0123-3210";
exDB[3] = "浅野";
exDB[4] = "埼玉";
exDB[5] = "1234-1234";
・・・・
exDB[27] = "山田";
exDB[28] = "神奈川";
exDB[29] = "9753-9753";
```

OTHERS

DETAILS

このサンプルは「13-1　外部ファイルからリストを作成する」で解説したものをさらに改良したものです。このサンプルではフォームを配置し、そこでキーワードを入力してもらい、そのキーワードを外部JavaScriptファイル内の文字列と比較して検索します。そして、検索結果を新たに開いたウィンドウに表示します。

外部配列の定義や読み込み方法はP.383のサンプルと同じです。詳しい説明はそちらを参照してください。

まずフォームを用意します。①のようにフォームはテキスト入力フィールドとボタンを配置します。ボタンにはonClickイベントハンドラを設定し、クリックされたらテキスト入力フィールドに入力された文字列を引数にして、dataSearch関数を呼び出すようにします。

呼び出されたdataSearch関数は引数を変数「ichar」に格納します。dataSearch関数では、最初に②の記述でicharが空文字でないかを調べ、空文字ならばアラートを出力後、カーソルのフォーカスをテキスト入力フィールドに設定し、dataSearch関数から抜けます。

空文字でないかを調べ終わったら、次に検索した結果、該当する文字があったかなかったかを保存する変数「flag」を用意します（③の箇所）。このflagの値は最初「false」に設定しておき、検索した結果、該当する文字列があったところで「true」に設定します。ここまで終わったら検索結果を表示する「resultWin」ウィンドウを開きます。

次は検索する部分です。検索は④のように引数として受け取った変数icharとexDB[i]を比較していきます。カウンタiはfor文の継続処理に「i+=cols」と設定されているため、3ずつ増加していきます。つまりexDB配列の添え字は3の倍数となるわけですが、今回はカラムが3つあるのでそうしてあります。同じだった場合には該当した配列要素から3つ分をresultWinウィンドウに出力し、⑤のように変数flagをtrueにします。この処理を配列の要素の個数まで繰り返します。

この文字列を比較する処理が終わった後で、変数flagがfalseだったならば、該当する文字列が配列内にないとして、⑥のように、アラートを表示する処理を行います。

ここまでで検索および表示処理が終わったので、⑦でresultWinウィンドウのdocumentオブジェクトを閉じます。この処理を行わないと動作が不安定になったりエラーになったりします。このスクリプトではこれ以外にprintData関数を用意して、Webページを読み込んだときに一覧表を作成する処理も行っています。

このようなスクリプトで、簡易データベースを実現することができます。この

データベースは一度ファイルをダウンロードしてしまえば、Webサーバーとコネクションすることなしに検索をかけることができます。ただし、この処理は比較的重いので、あまり大きなデータを扱うのには向いていません。

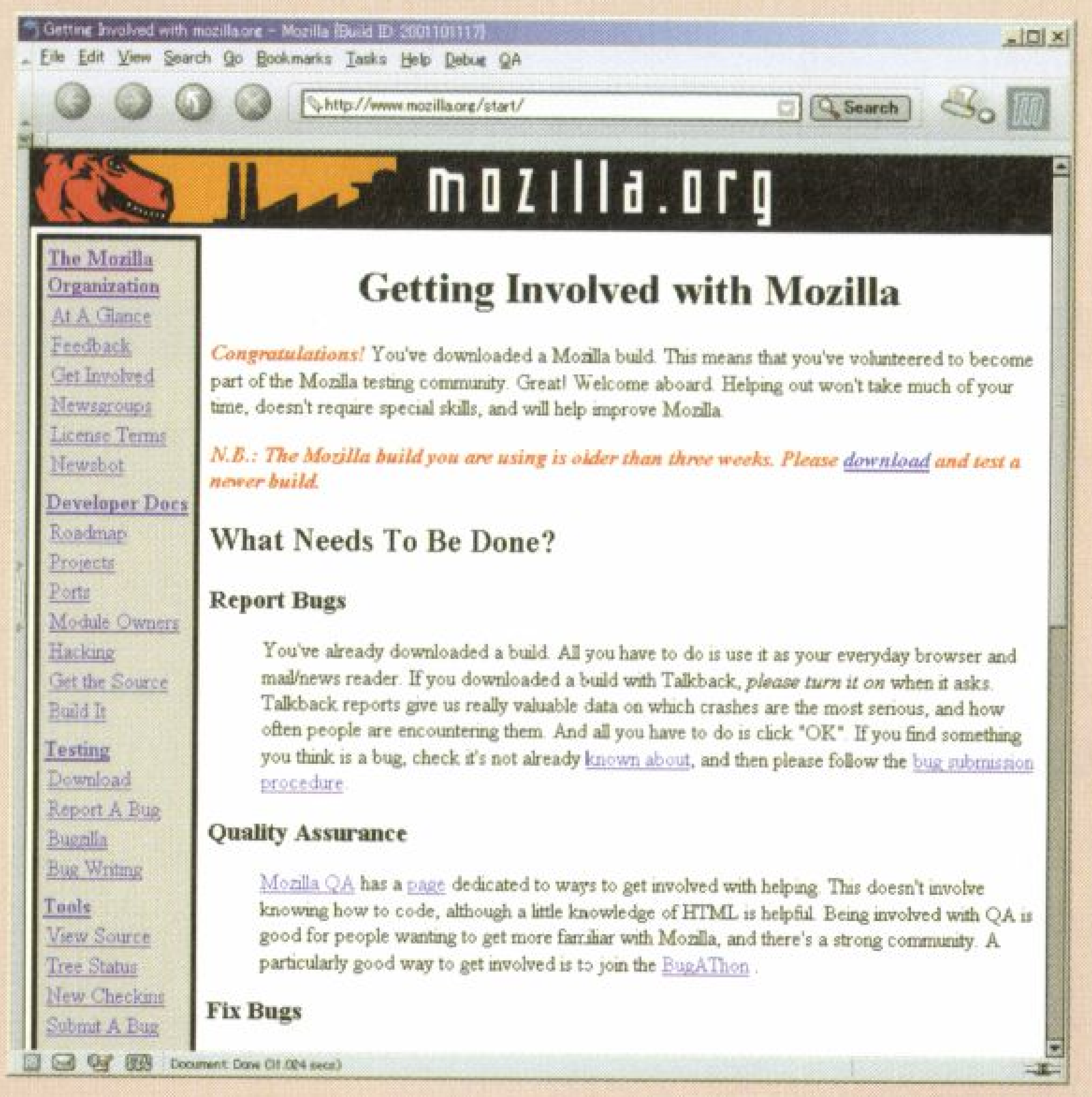
COLUMN

オープンソース

最近、オープンソースという言葉をよく聞きます。オープンソースにより開発されている代表的なソフトウェアといえばLinuxです。ご存じのようにLinuxは、さまざまな人の協力により、急速に改良が加えられているUNIX系のOSです。

そして、Mozillaもオープンソースにより開発されています。そう、NN6のコアとなったソフトウェアです。2001年11月現在、Mozilla0.9.5と呼ばれるバージョンになり、出荷は間近という状態です。

ただし、Mozillaの開発はオープンソースとしてあまりうまくいっているとはいえません。コミュニティの形成がうまくいっていないため、開発がなかなか進まないようです。とはいえ、革新的なHTMLレンダリングエンジンや、W3C準拠のDOMなどの実装は進んでいます。見た目は、次のようにNN6とほとんど同じです。



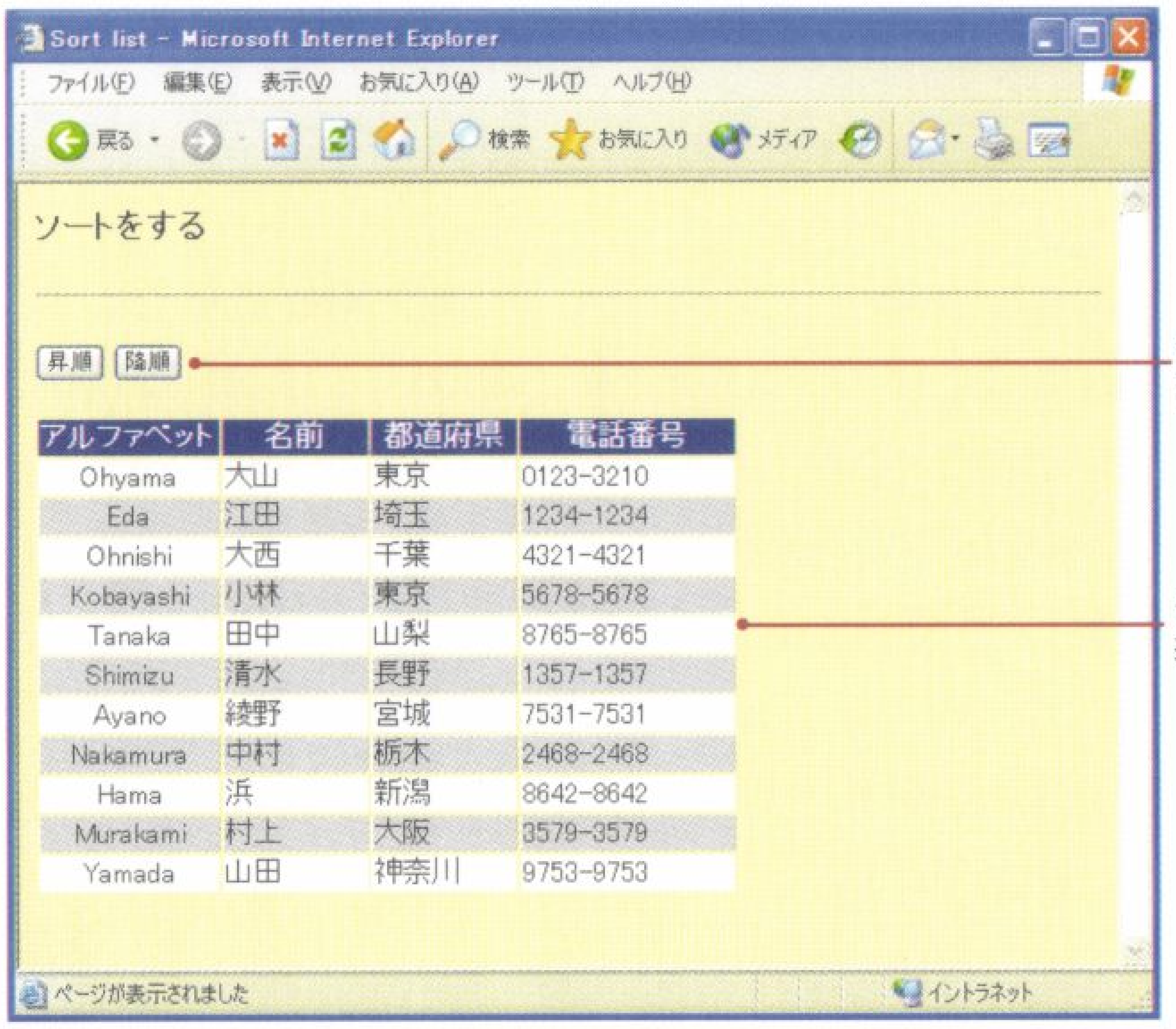
オープンソースとして開発を行おうとしているソフトウェアがいくつかありますが、成功の鍵はコミュニティの形成といった部分にありそうです。

13-3 WIN MAC

データをソートする

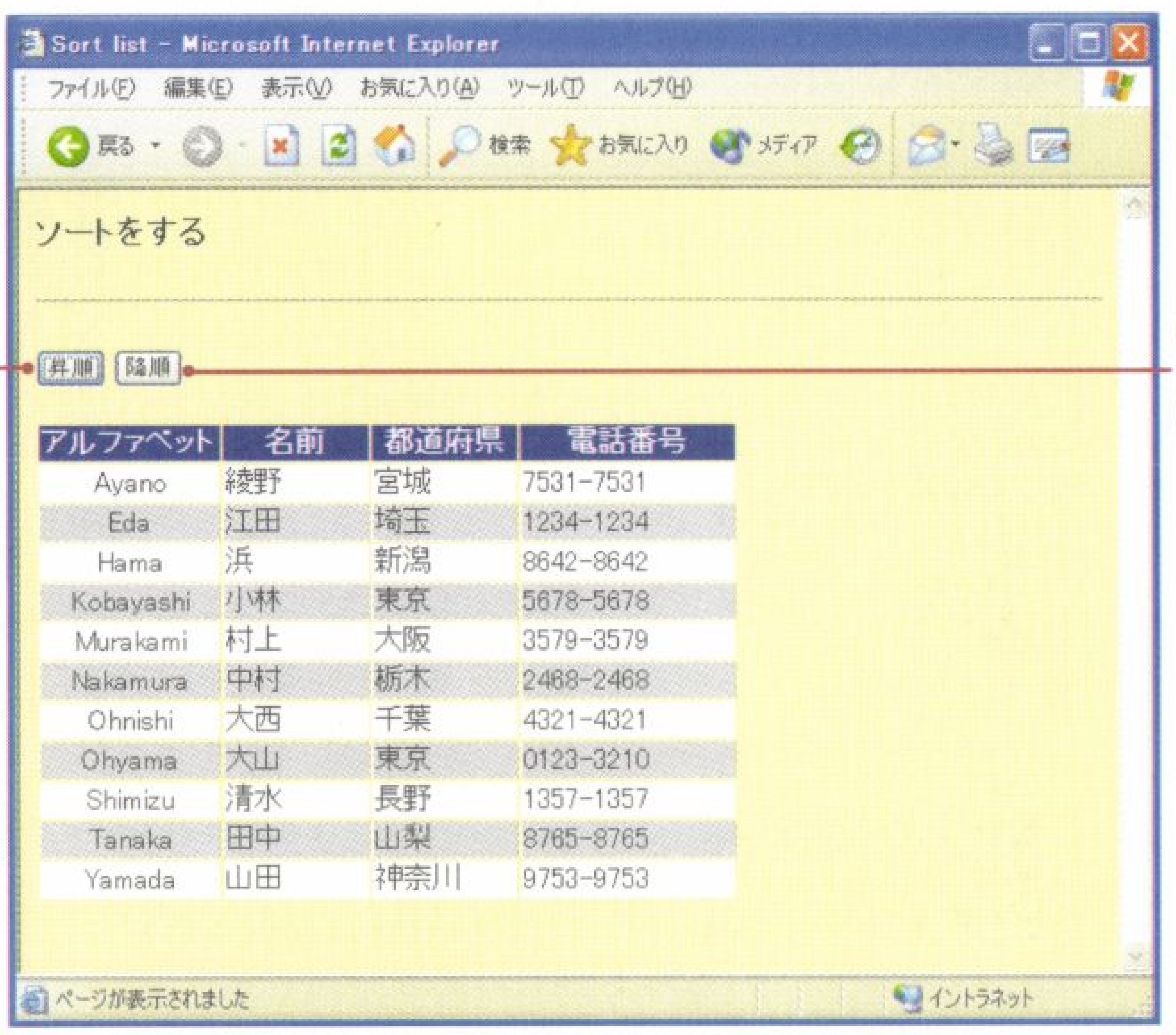
外部ファイルに書き込んである住所を読み込んできて、その中身をソートして表示することができるスクリプトです。一覧表などでは、目的のものを探すのに検索やソートなどの機能は非常に重要な機能です。JavaScriptでも当然組み込むことができます。

DISPLAY



ここをクリックすると…

データの一覧が表示される



データが昇順にソートされる

こちらをクリックすると、降順にソートする

SOURCE

sort.html

```
<html>
<title>Sort list</title>
<style type="text/css">
<!--
th {  background: #000080;
      color: #ffffff }
.light-center { background: #ffffff;
      text-align: center }
.light { background: #ffffff }
.dark-center { background: #dddddd;
      text-align: center }
.dark { background: #dddddd }
-->
</style>
<script language="javascript" src="./data2.js"></script> ―①
<script language="javascript">
<!--
var i;
var arrayLength = exDB.length; ―⑦

function printData(sortOrder){
   var msg;
   var data = new Array(); ―⑤

   if(sortOrder=="asc"){ ―②
      exDB.sort();
   }
   else if(sortOrder=="desc"){
      exDB.sort().reverse(); ―③
   }

   msg = "<table border='0'>"; ―⑥

   msg = msg + "<tr><th width='100'>アルファベット</th>
➡     <th widht='80'>名前</th><th width='80'>都道府県</th>
➡     <th width='120'>電話番号</th></tr>";

   for( i=0 ; i<arrayLength ; i++ ){
```

▼

```
data = exDB[i].split(","); ————④

        if( i%2 == 0 ){
            msg = msg + "<tr><td class='light-center'
⇒               width='100'>" + data[0] + "</td>";
            msg = msg + "<td class='light' width='80'>" + data[1]
⇒               + "</td>";
            msg = msg + "<td class='light' width='80'>" + data[2]
⇒               + "</td>";
            msg = msg + "<td class='light' width='120'>" + data[3]
⇒               + "</td></tr>";

        }
        else{
            msg = msg + "<tr><td class='dark-center'
⇒               width='100'>" + data[0] + "</td>";
            msg = msg + "<td class='dark' width='80'>" + data[1]
⇒               + "</td>";
            msg = msg + "<td class='dark' width='80'>" + data[2]
⇒               + "</td>";
            msg = msg + "<td class='dark' width='120'>" + data[3]
⇒               + "</td></tr>";
        }
    }
    msg = msg + "</table>";
    document.getElementById("displayPane").innerHTML = msg;
}
//-->
</script>
<body bgcolor="#fff8dc" onload="printData('')">
<h3>ソートをする</h3>
<hr />
<form name="myForm">
    <input type="button" value="昇順"
⇒       onclick="printData('asc')" />
    <input type="button" value="降順"
⇒       onclick="printData('desc')" />
</form>
```

```
<div id="displayPane"></div>
</body>
</html>
```

data2.js

```
var exDB = new Array();
exDB[0] = "Ohyama,大山,東京,0123-3210";
exDB[1] = "Eda,江田,埼玉,1234-1234";
exDB[2] = "Ohnishi,大西,千葉,4321-4321";
exDB[3] = "Kobayashi,小林,東京,5678-5678";
exDB[4] = "Tanaka,田中,山梨,8765-8765";
exDB[5] = "Shimizu,清水,長野,1357-1357";
exDB[6] = "Ayano,綾野,宮城,7531-7531";
exDB[7] = "Nakamura,中村,栃木,2468-2468";
exDB[8] = "Hama,浜,新潟,8642-8642";
exDB[9] = "Murakami,村上,大阪,3579-3579";
exDB[10] = "Yamada,山田,神奈川,9753-9753";
```

DETAILS

このスクリプトでは、外部ファイルに用意されている住所のリストを、Webページ上でソートするスクリプトです。

このスクリプトで読み込むファイルはdata2.jsというファイルになっており、配列の1つの要素内にアルファベット、名前、都道府県、電話番号が「,」(カンマ)を区切り文字として1つになって入っています。これを①の部分で<script>タグにより読み込んでいます。data2.js内では先頭で配列の宣言を行い、その後配列の要素に値を代入しています。

表を表示しているのは、printData関数です。この関数はWebページがロードされたときのonloadイベントハンドラ、[昇順]ボタン、[降順]ボタンをクリックしたときのonclickイベントハンドラにより呼び出されます。この関数ではアルファベットにより並び替えを行います。

printData関数は、引数を取ります。この引数sortOrderによりどのようにソートをするのかを決めます。printData関数ではまず②のように引数sortOrderに与えられた値の内容を確認します。「asc」が与えられている場合に

OTHERS

は昇順に、「desc」が与えられている場合には降順に、それ以外の場合にはとくに並び替えの処理を行いません。

並び替えにはsortメソッドとreverseメソッドを用いています。昇順の場合には単純のexDB配列をソートします。降順の場合には一度昇順に並べなおしてから、再度降順に並べなおしているので③のような記述になっています。

並べなおしたら後は表を表示するだけです。しかし、今回のexDB配列の要素は「,」で複数の値が連結された形になっています。そのため、splitメソッドを使って値を分割します(④)。分割された結果は配列に格納されるのでdata配列を作成し(⑤)、それに分割結果を代入しています。

配列データには以下のようにデータが格納されます。

配列データ	内容
data[0]	アルファベット
data[1]	名前
data[2]	都道府県
data[3]	電話番号

ここまで分割できたら、変数msgに表示したい内容を連結していきます。⑥の箇所から始まっていますが、この表でWebページに出力したい部分はすべて変数msgに格納されていきます。

表の作成はfor文が用いられています。今回for文はexDB配列の要素の数と同じだけ処理を繰り返します。このために⑦でexDB配列の個数をlengthプロパティを用いて取得しています。

for文中では、見た目のために2で割り切れる行と割り切れない行で背景色が変わるように処理をしています。スタイルシートで定義されたクラスをそれぞれが異なることで表を見やすくしています。

最後に、Web上に表示する部分です。Web上に表示したい部分はすべて変数msgに格納してきました。これをWebページ上に配置されている<div>要素の部分に代入します。<div>要素の特定にはgetElementByIdメソッドを用いています。特定された要素のinnerHTMLプロパティに変数msgを代入することによりWebページ上に表を表示します。

このように比較的簡単にソートは実現できます。sortメソッドは昇順、降順、文字コード順などが指定できるので、実装しだいではさまざまなことを実現できます。

なお、Macintosh版IEでは、innerHTMLプロパティをサポートしていないので動作しません。

Section 14 APPENDIX

A-1

オブジェクト一覧

オブジェクトとそれに関連付けられたメソッド、プロパティを紹介しておきます。ここで紹介するものは、JavaScript1.3までの規格の中から、よく使われるものを中心に取り上げています。

各メソッドやプロパティは、表の書式の中に記載しています。なお、書式の「obj」、「Window」は省略することができます。

anchorオブジェクト

Webページ中のリンクを扱うオブジェクトです。

●プロパティ

書式	内容
objAnchor.length	ドキュメント内のアンカーの数を返す

appletオブジェクト

JavaAppletを扱うオブジェクトです。

●プロパティ

書式	内容
objApplet.length	ドキュメント内のアプレットの数を返す

argumentオブジェクト

関数の引数を扱うオブジェクトです。

●プロパティ

書式	内容
objArgument.length	関数の引数の数を表す

arrayオブジェクト

配列要素を扱うオブジェクトです。

●メソッド

書式	内容
objArray.concat(objArray1, objArray2, …, objArrayN)	配列を連結する
objArray.join(char)	配列要素をCharで連結した文字列を返す
objArray.pop()	配列の末尾の要素を返す
objArray.push(element Data)	配列の末尾に要素を追加する
objArray.reverse()	配列の順序を反転する
objArray.shift()	先頭の要素を削除する
objArray.slice(start, end)	配列要素を抜き出す
objArray.sort(CompareFunction)	配列要素を並び替える
objArray.splice(index1,element1)	配列要素を置換する
objArray.unishift(element Data)	配列の先頭に要素を追加する

●プロパティ

書式	内容
objArray.length	配列の個数を返す、設定する

Dateオブジェクト

日付・時刻情報を扱うオブジェクトです。

●メソッド

書式	内容
objDate.getDate()	日付を取得する
objDate.getDay()	曜日を取得する
objDate.getFullYear()	西暦（4桁）を取得する
objDate.getHours()	時を取得する
objDate.getMilliseconds()	ミリ秒を取得する
objDate.getMinutes()	分を取得する
objDate.getMonth()	月を取得する
objDate.getSeconds()	秒を取得する
objDate.getTime()	1970/1/1 00:00:00からのミリ秒を取得する
objDate.getTimezoneOffset()	世界協定時からのずれを分で取得する
objDate.getUTCDate()	世界協定時の日付を取得する
objDate.getUTCDay()	世界協定時の曜日を取得する
objDate.getUTCFullYear()	世界協定時の西暦（4桁）を取得する
objDate.getUTCHours()	世界協定時の時を取得する

書式	内容
objDate.getUTCMilliseconds()	世界協定時のミリ秒を取得する
objDate.getUTCMinutes()	世界協定時の分を取得する
objDate.getUTCMonth()	世界協定時の月を取得する
objDate.getUTCSeconds()	世界協定時の秒を取得する
objDate.getUTCYear()	世界協定時の西暦（2桁）を取得する
objDate.getYear()	西暦（2桁）を取得する
objDate.parse（date)	1970/1/1 00:00:00からのミリ秒を取得する
objDate.setDate(date)	日付をセットする
objDate.setFullYear(year)	西暦（4桁）をセットする
objDate.setHours(hour)	時をセットする
objDate.setMilliseconds (milliseconds)	ミリ秒をセットする
objDate.setMinutes(minutes)	分をセットする
objDate.setMonth(month)	月をセットする
objDate.setTime(time)	時間を設定する
objDate.setUTCDate(date)	世界協定時の日付をセットする
objDate.setUTCFullYear(year)	世界協定時の西暦（4桁）をセットする
objDate.setUTCMilliseconds (milliseconds)	世界協定時のミリ秒をセットする
objDate.setUTCMinutes(minute)	世界協定時の分をセットする
objDate.setUTCMonth(month)	世界協定時の月をセットする
objDate.setUTCSeconds(seconds)	世界協定時の秒をセットする
objDate.setYear(year)	西暦（2桁）をセットする
objDate.toGMTString	グリニッジ標準時を文字列で取得する
objDate.toLocaleString	ローカル時を文字列で取得する
objDate.toUTCString	世界協定時を文字列で取得する
objDate.UTC(year,month,day [,hrs,hrs,min,sec,ms])	1970/1/1 00:00:00から指定時刻までのミリ秒を取得する

Documentオブジェクト

<body>～</body>による表示領域の操作を行うオブジェクトです

●メソッド

書式	内容
document.clear()	ドキュメントをクリアする
document.close()	ドキュメントを閉じる
document.open([mimeType])	ドキュメントを開く
document.write(string)	ドキュメントに出力する
document.writeln(string)	ドキュメントに出力する

●プロパティ

書式	内容
document.alinkColor	アクティブリンクの色を参照、設定する
document.bgColor	背景色の参照、設定をする
document.cookie	クッキー情報の参照、設定をする
document.domain	ドメイン名の参照、設定をする
document.fgColor	文字色の参照、設定をする
document.lastModified	最終更新日の参照をする
document.linkColor	リンクの色の参照、設定をする
document.location	ドキュメントの位置の参照、設定をする
document.referrer	1つ前のURLの参照をする
document.title	タイトルの参照、設定をする
document.URL	文書のURLの参照、設定をする
document.vlinkColor	リンク済みの色の参照、設定をする

eventオブジェクト

イベントを扱うオブジェクトです。

●プロパティ

書式	内容
event.height	イベントの起こった高さを取得する
event.keyCode	イベントを起こしたキーのUnicodeを参照する
event.modifiers	イベントを起こした修飾キーを参照する
event.type	イベントタイプを取得する
event.which	イベントを起こしたキーのISO-Latin-1コードを参照する
event.width	イベントの起こった幅を取得する
event.x	イベントのX座標を取得する
event.y	イベントのY座標を取得する

Element（Button）オブジェクト

<input type="button">によるボタンを扱うオブジェクトです。

●メソッド

書式	内容
document.objForm.objButton.blur()	フォーカスを外す
document.objForm.objButton.click()	クリックをする
document.objForn.objButton.focus()	フォーカスをする

APPENDIX

●プロパティ

書式	内容
document.objForm.objButton.name	フォームの名前を参照する
document.objForm.objButton.type	フォームのタイプを取得する
document.objForm.objButton.value	フォームの値を取得する

Element（Checkbox）オブジェクト

<input type="checkbox">によるチェックボックスを扱うオブジェクトです。

●メソッド

書式	内容
document.objForm.objCheckbox.blur()	フォーカスを外す
document.objForm.objCheckbox.focus()	フォーカスをする
document.objForm.objCheckbox.click()	クリックをする

●プロパティ

書式	内容
document.objForm.objCheckbox.checked	状態を参照、設定する
document.objForm.objCheckbox.defaultChecked	初期状態を参照、設定する
document.objForm.objCheckbox.name	フォームの名前を参照する
document.objForm.objCheckbox.type	フォームのタイプを取得する
document.objForm.objCheckbox.value	フォームの値を取得する

Element（Hidden）

<input type="hidden">による隠し要素を扱うオブジェクトです。

●プロパティ

書式	内容
document.objForm.objHidden.name	フォームの名前を参照する
document.objForm.objHidden.type	フォームのタイプを取得する
document.objForm.objHidden.value	フォームの値を取得する

Element（Option）オブジェクト

<option>による選択項目を扱うオブジェクトです。

●プロパティ

書式	内容
document.objForm.objOption.defaultSelected	初期状態を参照、設定する
document.objForm.objOption.index	参照番号を参照する
document.objForm.objOption.length	オプションの個数を参照する
document.objForm.objOption.selected	選択状態の参照をする
document.objForm.objOption.text	テキストを参照、設定する
document.objForm.objOption.value	フォームの値の参照、設定する

Element（Password）オブジェクト

<input type="passwd">によるパスワードフォームを扱うオブジェクトです。

●メソッド

書式	内容
document.objForm.objPassword.blur()	フォーカスを外す
document.objForm.objPassword.focus()	フォーカスをする
document.objForm.objPassword.select()	テキストの選択をする

●プロパティ

書式	内容
document.objForm.objPassword.defaultValue	ディフォルト文字の参照、設定する
document.objForm.objPassword.name	フォームの名前を参照する
document.objForm.objPassword.type	フォームのタイプを取得する
document.objForm.objPassword.value	フォームの値を取得する

Element（Radio）オブジェクト

<input type="radio">によるラジオボタンを扱うオブジェクトです。

●メソッド

書式	内容
document.objForm.objRadio.blur()	フォーカスを外す
document.objForm.objRadio.click()	テキストの選択をする
document.objForm.objRadio.focus()	フォーカスをする

●プロパティ

書式	内容
document.objForm.objRadio.checked	状態を参照、設定する
document.objForm.objRadio.defaultChecked	初期状態を参照、設定する
document.objForm.objRadio.length	オプションの個数を参照する
document.objForm.objRadio.name	フォームの名前を参照する
document.objForm.objRadio.type	フォームのタイプを取得する
document.objForm.objRadio.value	フォームの値を取得する

Element（Reset）オブジェクト

<input type="reset">によるリセットボタンを扱うオブジェクトです。

●メソッド

書式	内容
document.objForm.objReset.blur()	フォーカスを外す
document.objForm.objReset.click()	クリックをする
document.objForm.objReset.focus()	フォーカスをする

●プロパティ

書式	内容
document.objForm.objReset.name	フォーム名の名前を参照する
document.objForm.objReset.type	フォームのタイプを取得する
document.objForm.objReset.value	フォームの値を取得する

Element（Select）オブジェクト

<select>～</select>による選択フォームを扱うオブジェクトです。

●メソッド

書式	内容
document.objForm.objSelect.blur()	フォーカスを外す
document.objForm.objSelect.focus()	フォーカスをする

●プロパティ

書式	内容
document.objForm.objSelect.length	選択肢の個数を参照する
document.objForm.objSelect.name	フォームの名前を参照する
document.objForm.objSelect.options[N]	N+1番目の選択肢を参照する
document.objForm.objSelect.selectedIndex	選択されている選択の番号を参照する
document.objForm.objSelect.type	フォームのタイプを取得する
document.objForm.objSelect.value	フォームの値を取得する

Element（Submit）オブジェクト

<input type="submit">による送信ボタンを扱うオブジェクトです。

●メソッド

書式	内容
document.objForm.objSubmit.blur()	フォーカスを外す
document.objForm.objSubmit.click()	クリックをする
document.objForm.objSubmit.focus()	フォーカスをする

●プロパティ

書式	内容
document.objForm.objSubmit.name	フォームの名前を参照する
document.objForm.objSubmit.type	フォームのタイプを取得する
document.objForm.objSubmit.value	フォームの値を取得する

Element（Text）オブジェクト

<input type="text">による1行テキスト入力フォームを扱うオブジェクトです。

●メソッド

書式	内容
document.objForm.objText.blur()	フォーカスを外す
document.objForm.objText.focus()	フォーカスをする
document.objForm.objText.select()	クリックをする

●プロパティ

書式	内容
document.objForm.objText.name	フォームの名前を参照する
document.objForm.objText.type	フォームのタイプを取得する
document.objForm.objText.value	フォームの値を取得する

Element（Textarea）オブジェクト

<textarea>～</textarea>で囲まれたテキストエリアを扱うオブジェクトです。

●メソッド

書式	内容
document.objForm.objTextarea.blur()	フォーカスを外す
document.objForm.objTextarea.focus()	フォーカスをする
document.objForm.objTextarea.select()	クリックをする

●プロパティ

書式	内容
document.objForm.objTextarea.defaultValue	デフォルト文字列の参照、設定をする
document.objForm.objTextarea.name	フォームの名前を参照する
document.objForm.objTextarea.type	フォームのタイプを取得する
document.objForm.objTextarea.value	フォームの値を取得する

Formオブジェクト

<form>~</form>によるフォームを扱うオブジェクトです。

●メソッド

書式	内容
document.objForm.reset()	フォームの内容をリセットする
document.objForm.submit()	フォームを送信する

●プロパティ

書式	内容
document.objForm.action	送信先の参照、設定をする
document.objForm.encoding	エンコード形式の参照、設定をする
document.objForm.length	フォーム内の要素の個数を参照する
document.objForm.method	送信形式の参照、設定をする
document.objForm.name	フォームの名前を参照する
document.objForm.target	送信後のURIの参照、設定をする

Frameオブジェクト

ウィンドウ内のフレームの情報を扱うオブジェクトです。

●メソッド

書式	内容
objWindow.objFrame.blur()	フォーカスを外す
objWindow.objFrame.clearInterval(intervalID)	タイマーを解除する
objWindow.objFrame.clearTimeout(timerID)	タイマーを解除する
objWindow.objFrame.focus()	フォーカスをする
objWindow.objFrame.setInterval(IntervalID)	タイマーを設定する
objWindow.objFrame.setTimeout(timerID)	タイマーを設定する

●プロパティ

書式	内容
objWindow.objFrame.length	フレームの個数を参照する
objWindow.objFrame.location	フレームのロケーションを参照、設定する
objWindow.objFrame.name	フレームの名前を参照する
objWindow.objFrame.parent	親要素を参照する
objWindow.objFrame.self	自分自身を参照する
top.objFrame	トップの要素を参照する

Historyオブジェクト

Webブラウザの持つ履歴情報について扱うオブジェクトです。

●メソッド

書式	内容
objWindow.History.back()	履歴を戻す
objWindow.History.forward()	履歴を進める
objWindow.History.go(N)	履歴をNだけ移動する

●プロパティ

書式	内容
objWindow.History.length	履歴の個数を参照する

Imageオブジェクト

画像に関する情報を扱うオブジェクトです。

●プロパティ

書式	内容
document.objImage.border	枠幅を参照する
document.objImage.complete	読み込み状態を参照する
document.objImage.height	高さを参照、設定する
document.objImage.hspace	文字との間隔（横）を参照する
document.objImage.length	画像の数を参照する
document.objImage.lowsrc	低解像度の画像URIを参照、設定する
document.objImage.name	画像の名前を参照する
document.objImage.src	画像URIを参照する
document.objImage.vspace	文字との間隔（縦）を参照する
document.objImage.width	画像の幅を参照、設定する

Linkオブジェクト

Webページ中のリンクの情報を扱うオブジェクトです。

●プロパティ

書式	内容
document.link[N].hash	アンカーを参照、設定する
document.link[N].host	ホストを参照、設定する
document.link[N].hostname	ホスト名を参照、設定する
document.link[N].href	リンクを参照、設定する
document.link.length	リンクの個数を参照する

書式	内容
document.link[N].pathname	パス名を参照する
document.link[N].port	ポートを参照する
document.link[N].protocol	プロトコルを参照する
document.link[N].search	CGIへの文字列を参照、設定する
document.link[N].target	ターゲットを参照、設定する

locationオブジェクト

場所（URL）に関する情報を扱うオブジェクトです。

●メソッド

書式	内容
location.reload()	再読み込みをする
location.replace(URI)	URIへ移動する

●プロパティ

書式	内容
location.hash	アンカーを参照、設定する
location.host	ホストを参照、設定する
location.hostname	ホスト名を参照、設定する
location.href	リンクを参照、設定する
location.pathname	パス名を参照する
location.port	ポートを参照する
location.protocol	プロトコルを参照する
location.search	CGIへの文字列を参照、設定する

Mathオブジェクト

数学的な処理を扱うオブジェクトです。

●メソッド

書式	内容
Math.abs(x)	絶対値を求める
Math.acos(x)	アークコサインを求める
Math.asin(x)	アークサインを求める
Math.atan(x)	アークタンジェントを求める
Math.atan2(x,y)	x、yから角度を求める
Math.ceil(x)	切り上げて整数にする

書式	内容
Math.cos(x)	コサインを求める
Math.exp(x)	対数を求める
Math.floor(x)	切り下げて整数にする
Math.log(x)	ログの値を求める
Math.max(x,y)	x、yの大きいほうを求める
Math.min(x,y)	x、yの小さいほうを求める
Math.pow(x,y)	xのyのべき乗数を求める
Math.random()	0から1の乱数を発生させる
Math.round(x)	四捨五入する
Math.sin(x)	サインを求める
Math.sqrt(x)	平方根を求める
Math.tan(x)	タンジェントを求める

●プロパティ

書式	内容
Math.E	自然対数の値（e=2.718）
Math.LN10	10の自然対数の値
Math.LN2	2の自然対数の値
Math.LOG10E	10を底とするeの自然対数の値
Math.LOG2E	2を底とするeの自然対数の値
Math.PI	円周率の値
Math.SQRT1_2	2の平方根の1/2の値
Math.SQRT2	2の平方根の値

MineTypeオブジェクト

プラグインのMIMEタイプを扱うオブジェクトです。

●プロパティ

書式	内容
navigator.mimeTypes[mimetype].description	プラグインの詳細を参照する
navigator.mimeTypes[mimetype].enablePlugin	プラグインの状態を参照する
navigator.mimeTypes.length	MimeTypeの数を参照する
navigator.mimeTypes[mimetype].suffixes	プラグインの拡張子を参照する
navigator.mimeTypes[mimetype].type	MIMEタイプの名前を参照する

navigatorオブジェクト

Webブラウザ固有の情報を扱うオブジェクトです。

●メソッド

書式	内容
navigator.javaEnabled()	Javaが動作するかを参照する

●プロパティ

書式	内容
navigator.appCodeName	コード名を参照する
navigator.appName	アプリケーション名を参照する
navigator.appVersion	バージョンを参照する
navigator.language	言語セットを参照する
navigator.platform	プラットフォームを参照する
navigator.userAgent	Webブラウザ情報を参照する

Numberオブジェクト

数値を扱うオブジェクトです。

●プロパティ

書式	内容
Number.MAX_VALUE	最大値
Number.MIN_VALUE	最小値
Number.NaN	数値以外
Number.NEGATIVE_INFINITY	負の無限大
Number.POSITIVE_INFINITY	正の無限大

Pluginオブジェクト

プラグインを扱うオブジェクトです。

●メソッド

書式	内容
navigator.plugins[index].refresh()	プラグインを読み込み直す

●プロパティ

書式	内容
navigator.plugins[index].description	プラグインの詳細を参照する
navigator.plugins[index].filename	プラグインのファイル名を参照する
navigator.plugins.length	プラグインの個数を参照する
navigator.plugins[index].type	プラグインのタイプを参照する

RegExpオブジェクト

検索パターンを扱うオブジェクトです。

●メソッド

書式	内容
objRegExp.compile(pattern[,flag])	パターンを設定する
objRegExp.exec(str)	検索を実行する
objRegExp.test()	パターンの確認をする

●プロパティ

書式	内容
objRegExp.$1~$9	パターンと一致している部分を参照する
objRegExp.$*	改行コードの設定をする
objRegExp.$_	検索文字の指定をする
objRegExp.global	検索を繰り返して行うかどうかを参照する
objRegExp.ingnoreCase	大文字と小文字を区別するかどうかを参照する
objRegExp.input	検索文字の指定をする
objRegExp.lastIndex	検索開始位置を参照、設定する
objRegExp.lastMatch	最後にマッチした位置を参照する
objRegExp.lastParen	最後にマッチしたパターンを参照する
objRegExp.leftContext	マッチした左側の文字列を参照する
objRegExp.multiline	改行コードを参照、設定する
objRegExp.rightContext	マッチした右側の文字列を参照する
objRegExp.source	パターン文字を参照する

screenオブジェクト

画面の表示に関する情報を扱うオブジェクトです。

●プロパティ

書式	内容
screen.availHeight	画面の有効な高さを参照する
screen.availWidth	画面の有効な幅を参照する
screen.colorDepth	画面の色深度を参照する
screen.height	画面の高さを参照する
screen.pixelDepth	オフスクリーンバッファの色深度を参照する
screen.width	画面の幅を参照する

Stringオブジェクト

文字列を処理するオブジェクトです。

●メソッド

書式	内容
objString.anchor(anchor)	アンカーを設定する
objString.big()	大きくする
objString.blink()	点滅させる
objString.bold()	太字にする
objString.charAt(N)	1文字抜き出す
objString.charCodeAt(N)	unicodeに変換する
objString.concat(objString2[, ...ObjStringN])	連結を行う
objString.find(pattern)	検索し、真理値を返す
objString.fixed()	固定幅フォントにする
objString.fontColor(color)	色を設定する
objString.fontsize(size)	フォントサイズを設定する
objString.fromCharCode(code1[,…codeN])	Unicodeを文字に変換する
objString.indexOf(pattern, startposition)	検索を左から行う
objString.italics()	イタリックにする
objString.lastIndexOf(pattern, fromposition)	検索を右から行う
objString.link(URI)	リンクを設定する
objString.match(pattern)	検索し、文字列を返す
objString.replace(pattern, str)	置換を行う
objString.search(pattern)	検索を行い、先頭位置を返す
objString.slice(start[, end])	文字列を切り出す
objString.small()	小さくする
objString.split([separator][,limit])	指定文字で分割する
objString.strike()	打ち消し線を付ける
objString.sub()	下付き文字にする

書式	内容
objString.substr(start[,length])	指定位置から文字列を切り出す
objString.substring(startindex, endindex)	指定範囲の文字列を切り出す
objString.sup()	上付き文字にする
objString.toLowerCase()	小文字に変換する
objString.toUpperCase()	大文字に変換する

●プロパティ

書式	内容
objString.length	文字列の長さを参照する

windowオブジェクト

最上層のオブジェクトで、多くのオブジェクトを含みます。Webブラウザそのものや、ダイアログボックスなどを扱うオブジェクトです。

●メソッド

書式	内容
objWindow.alert(str)	アラートを表示する
objWindow.blur()	フォーカスを外す
objWindow.clearInterval(intervalID)	タイマーを解除する
objWindow.clearTimeout(timerID)	タイマーを解除する
objWindow.close()	ウィンドウを閉じる
objWindow.confirm(str)	確認ダイアログを表示する
objWindow.focus()	フォーカスをあてる
objWindow.home()	ホームに戻る
objWindow.moveBy(h, v)	ウィンドウをh,vだけ移動する
objWindow.moveTo(x,y)	ウィンドウをx,yに移動する
objWindow.open("windowname"[,options])	ウィンドウを開く
objWindow.prompt(str[,inputDefault])	プロンプトを表示する
objWindow.resizeBy(h, v)	ウィンドウを相対サイズで変更する
objWindow.resizeTo(width, height)	ウィンドウを絶対サイズで変更する
objWindow.scroll(x,y)	絶対位置にスクロールする
objWindow.scrollBy(h, v)	相対位置にスクロールする
objWindow.scrollTo(x, y)	絶対位置にスクロールする
objWindow.setInterval(intervalID)	タイマーを設定する
objWindow.setTimeout(timerID)	タイマーを設定する

●プロパティ

書式	内容
objWindow.closed	ウィンドウの開閉状態を参照する
objWindow.defaultStatus	初期状態のステータスバーを参照、設定する
objWindow.innerHeight	ウィンドウの内側の高さを参照、設定する
objWindow.innerWidth	ウィンドウの内側の幅を参照、設定する
objWindow.length	ウィンドウの数を参照する
objWindow.name	ウィンドウの名前を参照する
objWindow.opener	開いたウィンドウを参照する
objWindow.outerHeight	ウィンドウの外側の高さを参照、設定する
objWindow.outerWidth	ウィンドウの外側の幅を参照、設定する
objWindow.parent	親ウィンドウを参照する
objWindow.status	ステータスバーを参照、設定する

その他のメソッド、プロパティ

次に紹介するメソッドとプロパティは、すべてのオブジェクトで利用できるものです。

●メソッド

書式	内容
escape(str)	ISO Latin 1コードに変換する
eval(str)	実行結果を取り出す
isFinite(number)	有限かどうか調べる
isNaN(value)	数値かどうか調べる
parseFloat(string)	浮動小数点以下を取り出す
parseInt(string)	整数を取り出す
obj.toString()	文字列に変換する
unescape(str)	ISO Latin 1コードを文字に変換する
void()	何もしない

●プロパティ

書式	内容
Infinity	正の無限大を参照する
NaN	数値でない

A-2

JavaScriptでの色の指定方法

色は、Webページを制作する際には非常に重要なものです。しかし、あまり見たことのない記述方法を使ったり、WebブラウザやOSによって、少しずつ表現される色が異なったりするため、その扱いはちょっと面倒なものです。たとえば、WindowsとMacintoshでは、カラーマップが異なるため若干見え方が違います。JabaScriptに限らず、このOSによる表現の違いは、色を扱う上で必ずある問題であることを認識しておいてください。

JavaScriptでは、4種類の方法で色を扱うことができます。「色の名前で指定」、「RGB16進数で指定」、「RGB256進数で指定」、そして「16777216の数値で指定」という方法です。一番理解しやすいのは色の名前で指定する方法です。

```
document.bgColor="色の名前";
```

たとえば、Red(赤色)を指定する場合の記述例は、次のようになります。

```
document.bgColor="red";
```

これはIEでもNNでもどちらでも使うことができますが、一部の色はどちらかしか出なかったりすることもあります。

次にRGB16進数による方法です。これは色の3原色である赤(Red)、緑(Green)、青(Blue)で色を表現する方法です。それぞれR、G、Bを16進数(0からFまで)の数字2桁で表し、全部で6桁で表現します。10進数と16進数の変換は以下のようになります。

10進数	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16進数	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

この方法で表現できる色は16777216色です。記述は最初に「#」を付けて、次のように表現します。

```
#RRGGBB
```

このR、G、Bの部分がそれぞれ0からFまでの数字、アルファベットで表現されます。数値が大きければその要素の色が強いということになります。なお基準となる色は次のようになります。

RGB16進数	色名
#000000	Black（黒）
#FF0000	Red（赤）
#00FF00	Lime（ライム）
#0000FF	Blue（青）
#FFFFFF	White（白）

3つ目は、RGB256進数で表現する方法です。RGB16進数を256進数に変換したものと考えることもできます。RGB256進の場合には次のような記述で色を指定します。

```
rgb(R, G, B)
```

括弧内のR、G、Bはそれぞれ0から255までの数値です。rgb(0,0,0)は黒となり、rgb(255,255,255)が白になります。

最後に16777216の数値で表現する方法です。JavaScriptでは、「0」〜「16777216」で色を指定することができます。しかし、さすがにこんな膨大な数の色を覚えることはできません。そこで、RGB256進数で表現された色を変換して使うという方法が一般的です。RGB256進数から16777216の数値への変換は以下のように行います。このときR、G、Bはそれぞれ0から255の値です。

```
16777216の数値 = R×256×256 + G×256 + B
```

こういった指定の方法は、HTMLでの指定と同じです。実際に指定できる色名やRGB16進数、RGB256進数に関しての詳細は、ハンドブックシリーズの『HTMLハンドブック』や『Web Designer's Handbook』（共にソフトバンクパブリッシング刊）などを参照してください。

数字・記号

A

B・C

D

Top 196Page

か行

さ行

た行・な行

JavaScript Handbook 3rd edition

2001年11月30日　初版第1刷 発行
2002年11月 5日　初版第3刷 発行

著　者…………………宮坂（みやさか）　雅輝（まさてる）

発行者…………………稲葉　俊夫
発行所…………………ソフトバンク パブリッシング株式会社
〒107-0052　東京都港区赤坂4-13-13
電話　販売　03(5549)1200
編集　03(5549)1146
印　刷…………………クニメディア株式会社
組　版…………………横山浩一（C-Cable）

イラスト●陶延　千晴（C-Cable）
デザイン●阿部　岳（www.gakudesign.com）

乱丁本・落丁本は小社編集局にてお取り替えいたします。
定価はカバーに記載されております。
本書内容に関するご質問などは、ご面倒でも小社書籍編集部まで、必ず書面にてご連絡くださいますようにお願いいたします。

Printed in Japan　ISBN4-7973-1819-8

SOFT BANK
Publishing
ソフトバンク パブリッシング

SOFTBANK Publishing ソフトバンク パブリッシング

9784797318197

1920055018005

ISBN4-7973-1819-8

C0055 ¥1800E

定価 本体1,800円 ＋税